TS. NGUYỄN NGỌC GIANG (Chủ biên)
ThS. PHAN XUÂN VỌNG - NGUYỄN QUỐC ANH

ĐƯỜNG VÀO LẬP TRÌNH PYTHON

Một trong những ngôn ngữ lập trình phổ biến nhất của AI, Machine Learning, Data Mining, Deep Learning và chương trình giáo dục phổ thông môn tin học

ĐHQG Hà Nội
NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

TS. NGUYỄN NGỌC GIANG. Tiến sĩ chuyên ngành Lí luận & Phương pháp dạy học Bộ môn toán, Cao học chuyên ngành Khoa học máy tính, Cử nhân Ngữ Văn Anh. Anh là học sinh Khóa I chuyên tin Khối phổ thông chuyên toán - tin, Đại học Sư phạm Vinh, Nghệ An.

Anh hiện là giảng viên Trường Đại học Ngân hàng TP. Hồ Chí Minh; tác giả của hơn 40 Đầu sách về toán, tin, vật lí và tiếng Anh. Anh có hàng chục bài viết trên tạp chí Tin học & nhà trường. Ngoài ra vốn là người giàu cảm xúc, anh cũng là tác giả của cuốn thơ "Giọt gió treo cành nắng".

Một số cuốn sách tin học gây được tiếng vang trong cộng đồng là: Khám phá trong giải toán phổ thông bằng các phương pháp toán - tin; Sáng tạo trong toán lập trình, Tích hợp toán, tin và vật lí, ...

ThS. PHAN XUÂN VỌNG. Thạc sĩ chuyên ngành Khoa học máy tính, Cử nhân sư phạm toán. Anh là giáo viên trường chuyên Đại học Vinh, có nhiều năm kinh nghiệm chuyên dạy cho đội tuyển tham dự kì thi Học sinh giỏi quốc gia môn tin học. Anh cũng là học sinh Khóa I chuyên tin Khối phổ thông chuyên toán - tin, Đại học Sư phạm Vinh, Nghệ An.

NGUYỄN QUỐC ANH Cử nhân chuyên ngành Hệ thống Điện. Có nhiều nghiên cứu chuyên sâu về Công nghệ thông tin. Đã xuất bản một số bài báo cũng như sách về Toán – Tin và ứng dụng Tin học trong Toán học.

TS. NGUYỄN NGỌC GIANG (Chủ biên)
ThS. PHAN XUÂN VỌNG – NGUYỄN QUỐC ANH

ĐƯỜNG VÀO LẬP TRÌNH PYTHON

(Một trong những ngôn ngữ lập trình chính, phổ biến nhất của AI, Machine Learning, Data Mining, Deep Learning và môn Tin học trong chương trình Giáo dục phổ thông mới)

NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA HÀ NỘI

MỤC LỤC

LỜI NÓI ĐẦU 5

Chương 1. MỞ ĐẦU 7

Chủ đề 1. Giới thiệu về lập trình Python 7

Chủ đề 2. Cài đặt Python và các phần mềm liên quan 9

Chương 2. LẬP TRÌNH PYTHON CƠ BẢN 15

Chủ đề 3. Vào/ra dữ liệu từ thiết bị chuẩn 15

Chủ đề 4. Biến và gán giá trị cho biến 23

Chủ đề 5. Kiểu dữ liệu số 30

Chủ đề 6. Cấu trúc rẽ nhánh 41

Chủ đề 7. Cấu trúc lặp 58

Chủ đề 8. Kiểu dữ liệu chuỗi 75

Chủ đề 9. Một số thao tác trên chuỗi 92

Chủ đề 10. Kiểu dữ liệu danh sách - List 112

Chủ đề 11. Một số vấn đề nâng cao về danh sách 130

Chủ đề 12. Kiểu dữ liệu bộ - Tuple 154

Chủ đề 13. Kiểu dữ liệu tập hợp - Set 166

Chủ đề 14. Kiểu dữ liệu từ điển - Dict 181

Chủ đề 15. Kiểu dữ liệu tệp 194

Chương 3. MỘT SỐ CHỦ ĐỀ PYTHON NÂNG CAO 210

Chủ đề 16. Chương trình con 210

Chủ đề 17. Chương trình con đệ quy, đệ quy quay lui và đệ quy có nhớ 229

Chủ đề 18. Kiểu dữ liệu lớp - Class 251

Chủ đề 19. Modunles và Packages 266
Chủ đề 20. Modunle queue và cấu trúc dữ liệu hàng đợi, ngăn xếp, hàng đợi ưu tiên 277
Chủ đề 21. Xử lý ngoại lệ - Exceptions 310
Chủ đề 22. Đồ họa cùng với Tkinter 318

TÀI LIỆU THAM KHẢO 347

LỜI NÓI ĐẦU

Cuộc cách mạng công nghệ 4.0 đã và đang làm thay đổi mọi lĩnh vực khoa học và đời sống. Các ngành nghề dựa vào thành quả của lĩnh vực công nghệ cao như Công nghệ Nano, Công nghệ Sinh học và đặc biệt là Công nghệ Thông tin ngày càng phát triển vượt bậc cả về lượng lẫn về chất. Để góp phần cho ngành Công nghệ thông tin có ảnh hưởng mạnh mẽ như vậy, thì việc lựa chọn ngôn ngữ lập trình trong các lĩnh vực mũi nhọn như Trí tuệ nhân tạo (AI), học máy (Machine Learning), khai phá dữ liệu (Data Mining), học sâu (Deep Learning) trở nên vô cùng quan trọng và cần thiết. Một trong những ngôn ngữ đáp ứng được hầu hết các tiêu chí của tất cả nhà lập trình khó tính nhất đó chính là ngôn ngữ lập trình Python.

Ngôn ngữ lập trình Python có nhiều ưu điểm nổi trội như dễ nhớ, dễ viết, khả năng xử lí số liệu lớn, phức tạp rất tốt, thư viện có nhiều hàm, đáp ứng được nhiều kiểu dữ liệu mới của Machine Learning, AI, Data Mining, Deep Learning. Ngày nay, máy tính có khả năng tự học mà không cần phải lập trình một cách rõ ràng. Ngành Khoa học máy tính hiện có nhiều ứng dụng sâu rộng vào cuộc sống hằng ngày như đánh cờ, nhận diện khuôn mặt, chẩn đoán y khoa, phát hiện thẻ tín dụng giả, dự đoán kết quả trận đấu, nhận diện giọng nói, phân loại các chuẩn DNA, tóm tắt văn bản, trả lời tự động,... Chính vì thế, ngôn ngữ lập trình Python giờ đã trở thành một yếu tố không thể thiếu khi nhắc đến AI, Machine Learning, Data Mining, Deep Learning và ngược lại.

Ngôn ngữ lập trình Python vừa đáp ứng được yêu cầu của các bài toán lập trình cổ điển trước đây và các bài toán lập trình mới. Tuy nhiên, các tài liệu về lập trình Python ở nước ta còn thiếu. Đó là rào cản lớn cho những bạn muốn sử dụng ngôn ngữ này trong lập trình. Ngoài ra, sắp tới ngôn ngữ lập trình Python sẽ được đưa vào giảng dạy trong chương trình Giáo dục phổ thông môn Tin học nên việc biên

soạn tài liệu có chất lượng về Python cho các em học sinh là một việc làm rất cấp thiết. Cuốn sách "***Đường vào lập trình Python***" là sản phẩm "thai nghén" trong một thời gian dài vừa viết, vừa phản biện sao cho thành phẩm cuối cùng được ưng ý nhất. Cuốn sách có 22 chủ đề được viết theo lối viết sư phạm. Ngoài chương 1 là chương về kiến thức mở đầu ra thì các chương còn lại có cấu trúc chung như sau:

A. Đặt vấn đề

B. Bài tập ôn luyện

C. Thuật toán và hướng dẫn giải

Nội dung cuốn sách bàn về những vấn đề cơ bản của lập trình như cấu trúc lặp, kiểu dữ liệu chuỗi,... cũng như những vấn đề nâng cao, hiện đại như quay lui, hàng đợi, ngăn xếp, đồ họa, kiểu dữ liệu từ điển,... Chính vì thế, cuốn sách đáp ứng được mọi nhu cầu từ thấp đến cao, từ đơn giản đến phức tạp của tất cả các em học sinh Phổ thông Trung học, sinh viên, học viên cao học ngành Khoa học máy tính, quý thầy cô và nhà nghiên cứu muốn tìm hiểu về ngôn ngữ lập trình Python. Những ai quan tâm đến lập trình Python đều tìm thấy nhiều điều bổ ích từ cuốn sách này.

Mặc dù được biên soạn cẩn thận, tuy nhiên cuốn sách khó có thể tránh khỏi thiếu sót. Mọi ý kiến của quý độc giả về cuốn sách xin được gửi về cho chúng tôi theo địa chỉ: *Ông Nguyễn Ngọc Giang, Giảng viên Trường Đại học Ngân hàng Thành phố Hồ Chí Minh, 36 Tôn Thất Đạm, Phường Nguyễn Thái Bình, Quận 1, Thành phố Hồ Chí Minh. Điện thoại: 0908576218; Email: nguyenngocgiang.net@gmail.com.*

Chúng tôi trân quý mọi góp ý của quý bạn đọc và xin chân thành cảm ơn!

Thay mặt nhóm tác giả
TS. Nguyễn Ngọc Giang

Chương 1. MỞ ĐẦU

CHỦ ĐỀ 1. GIỚI THIỆU VỀ LẬP TRÌNH PYTHON

1. Tại sao lại là lập trình Python?

Ngôn ngữ lập trình Python được Guido van Rossum, lập trình viên nổi tiếng người Hà Lan viết, phiên bản đầu tiên được phát hành vào năm 1991. Hiện nay, Python đã phát triển đến các phiên bản 3.x.x. Python chạy được trên nhiều nền tảng khác nhau như Windows, Mac OS, Linux, Raspberry Pi,... Python được thiết kế với ưu điểm vượt trội là dễ đọc, dễ học và dễ nhớ; có cấu trúc lệnh đơn giản nhưng rất hiệu quả, phần lớn các câu lệnh được viết tương tự như ngôn ngữ tiếng Anh. Python còn cung cấp phong phú các thư viện, mô-đun cho phép làm việc thuận lợi về trí tuệ nhân tạo (Artificial Intelligence – AI), phân tích dữ liệu lớn (Big Data – BG), học máy (Machine Learning – ML), làm việc được hầu hết với các cơ sở dữ liệu. Triết lý căn bản khi thiết kế Python được nhóm thiết kế đưa ra:

"Đẹp đẽ tốt hơn xấu xí
Minh bạch tốt hơn che đậy
Đơn giản tốt hơn phức tạp..."

Chính vì có nhiều ưu điểm như trên, nên đã có nhiều kỹ sư phần mềm lựa chọn Python cho công việc của mình. Nhiều công ty đã sử dụng ngôn ngữ lập trình này để phát triển phần mềm như Google, Yahoo, CERN, NASA...

2. Sách này dành cho ai?

Mục tiêu của cuốn sách nhằm cung cấp một khóa học nhanh và đầy đủ nhất có thể cho bạn đọc về lập trình Python. Cuốn sách đề cập

từ những kiến thức cơ bản đến những vấn đề nâng cao. Cung cấp cho người học những vấn đề cốt lõi của Python, để từ đó có thể lập trình xử lý những công việc riêng của mình.

Cuốn sách được thiết kế theo định hướng chương trình phổ thông 2018, do vậy rất phù hợp với các em học sinh cấp trung học cơ sở, trung học phổ thông; học sinh chuyên tin; sinh viên năm nhất các trường cao đẳng, đại học chuyên ngành Công nghệ thông tin (CNTT). Ngoài ra, sách cũng rất phù hợp với các bạn muốn tự học lập trình cho mục đích riêng của mình.

3. Để học Python cần chuẩn bị những gì?

Để có thể tiến hành học, các bạn sẽ cần chuẩn bị:

+ Một máy vi tính có cài sẵn hệ điều hành (Linux, Mac OS hay Windows đều được).

+ Một trình biên dịch Python. (Trong nội dung cuốn sách này tác giả sử dụng phiên bản 3.7.2, 64 bit).

+ Một trình Text Editor (Geany, Visual Studio Code, Sublime Text...).

Việc cài đặt các công cụ liên quan trên từng hệ điều hành sẽ được hướng dẫn chi tiết cho bạn đọc ở phần kế tiếp của cuốn sách.

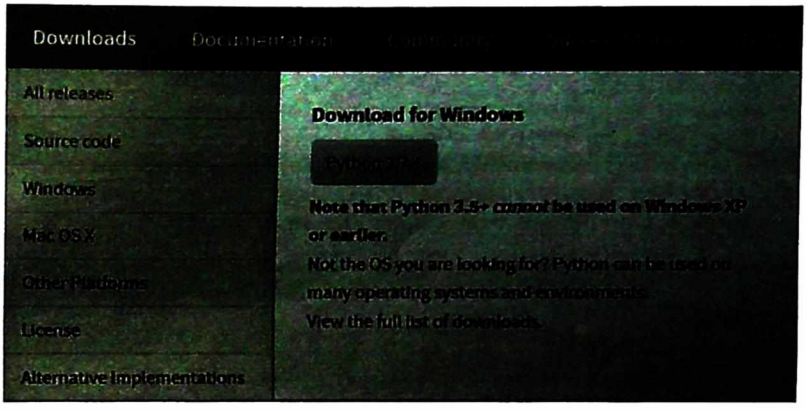
CHỦ ĐỀ 2. CÀI ĐẶT PYTHON VÀ CÁC PHẦN MỀM LIÊN QUAN

I. Cài đặt Python

1. Cài đặt Python trên Windows

Bước 1. Truy cập vào trang: https://www.python.org

Bước 2. Click vào nút Downloads, rồi nhấn vào Windows để tải file cài đặt Python.



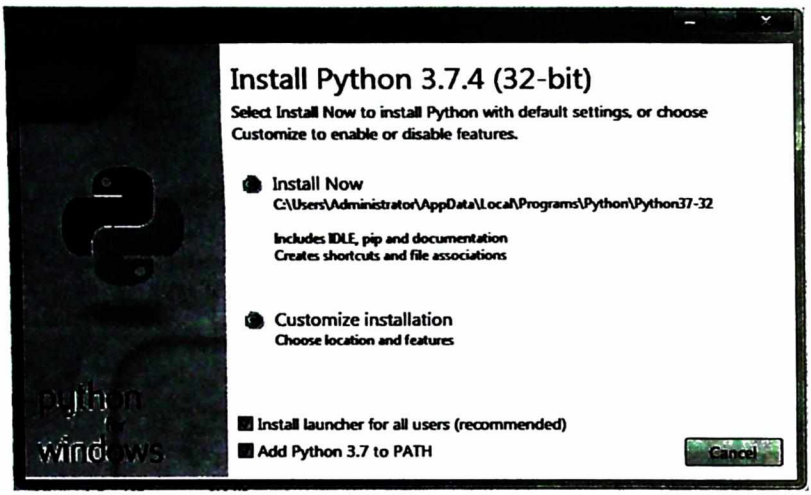
Bấm vào nút lệnh Python 3.7.4 để tải về, ta có tệp. python-3.7.4

Nếu muốn lựa chọn phiên bản khác, chọn View the full list of downloads.

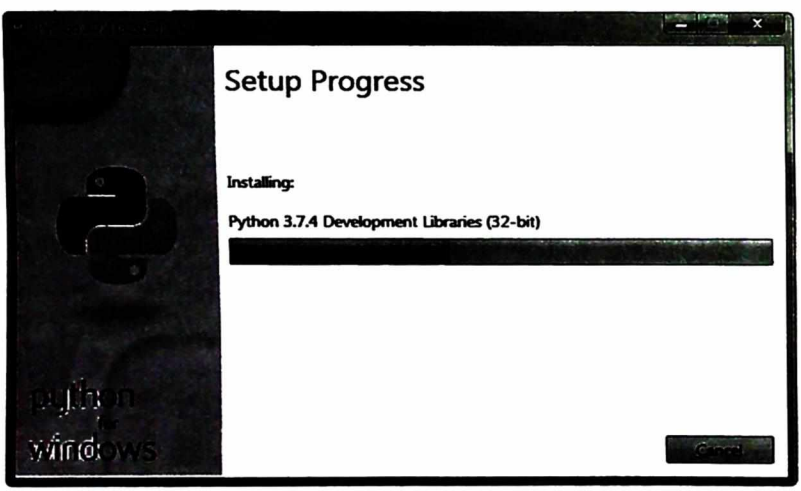
Ngoài ra, ta có thể tải bộ cài đặt ***excutable installer*** phù hợp với phiên bản Windows đang sử dụng (32/64 bit)

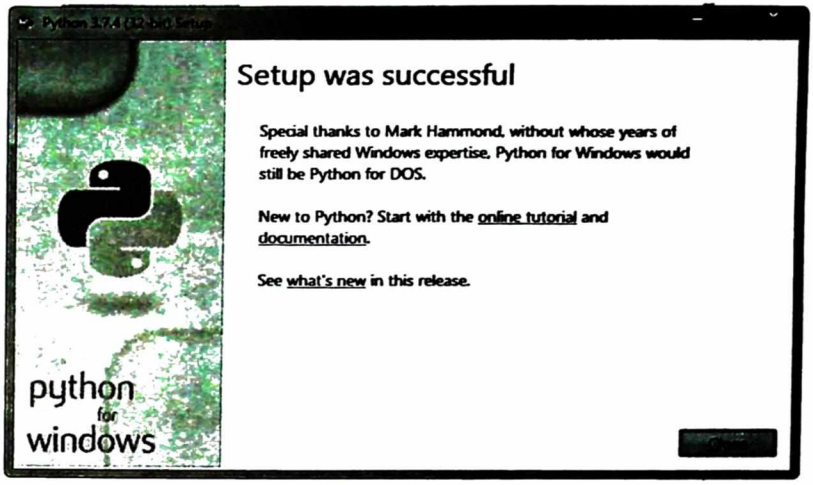
Bước 3. Cài đặt

- Nhấn đúp vào file vừa tải về.
- Tích vào ô: **Add Python 3.7 to PATH**
- Chọn **Install Now**



Bước 4. Chờ đợi quá trình cài đặt diễn ra, khoảng 30s.

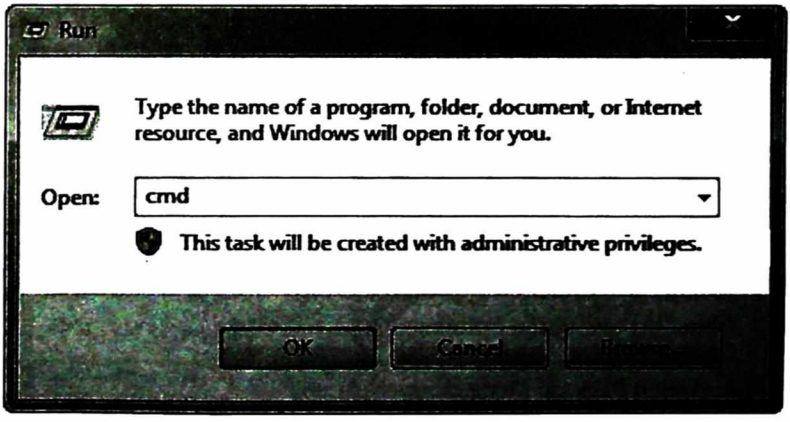




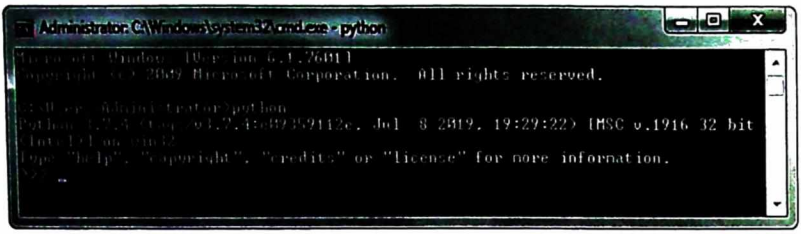
Nhấn **Close** để hoàn thành quá trình cài đặt.

Bước 5. Kiểm tra lại.

Vào cửa sổ lệnh Run và gõ **cmd**.



Nhấn **Enter** hoặc nhấn **OK,** một cửa sổ hiện ra, ta gõ **python** vào dấu nhắc lệnh sau đó nhấn tiếp **Enter**:



Kiểm tra

❖ ***Chú ý***

- Trong một số trường hợp báo lỗi, `'python' is not recognized as an internal or external command, operable program or batch file.` Đây là lỗi chưa đăng kí biến đường dẫn (**path variable**) cho **python.exe**. Lỗi này là do không chọn ô **Add Python 3.7 to PATH.** Trong trường hợp này, đăng kí lại biến môi trường (**Environment Variables**) hoặc cài lại và làm đầy đủ như hướng dẫn trên.

- Gõ lệnh **exit()** hoặc **Ctrl+D** để thoát lệnh **python3**.

- Việc cài đặt Python trên các hệ điều hành Mac OS và Linux được thực hiện tương tự.

II. Cài đặt một trình biên tập

1. Khái niệm "Trình biên tập"

Trình biên tập (Text Editor) là một phần mềm hỗ trợ soạn thảo, chỉnh sửa, biên tập và lưu trữ chương trình. Nội dung dưới đây sẽ đề cập tới một số **trình biên tập** dùng để biên tập, chỉnh sửa và hỗ trợ chạy (thực thi) mã nguồn Python.

2. Một số trình biên tập thông dụng

Hiện nay có rất nhiều những trình biên tập khác nhau bao gồm cả miễn phí và trả phí. Trong nội dung cuốn sách này, chúng ta sẽ đề cập đến hai trình biên tập hỗ trợ cả 3 nền tảng Windows, Linux và cả Mac OS, với rất nhiều tính năng thông dụng hiện nay đó là Sublime Text và Visual Studio Code.

3. Cài đặt Sublime Text

Bước 1. Truy cập địa chỉ: https://www.sublimetext.com/3

Chọn phiên bản phù hợp với hệ điều hành của bạn và tiến hành cài đặt.

Bước 2. Tùy chỉnh lại Sublime Text để có thể biên dịch được mã nguồn Python bằng cách: Tools → Build System → Python.

4. Cài đặt Visual Studio Code

Bước 1. Truy cập địa chỉ: https://code.visualstudio.com/download

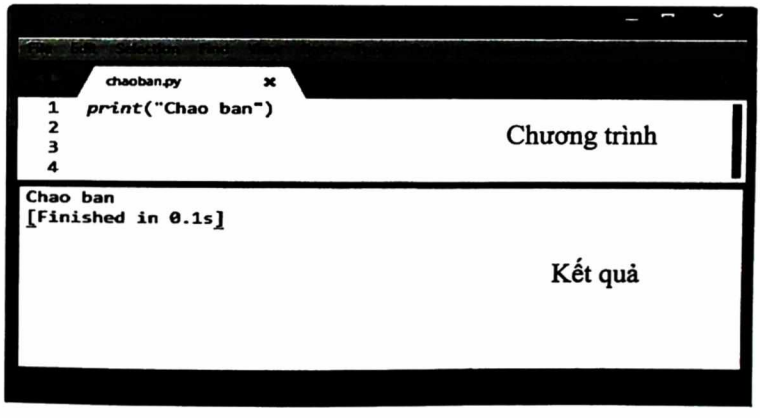
Chọn phiên bản phù hợp với hệ điều hành của bạn và tiến hành cài đặt.

Bước 2. Tùy chỉnh lại Visual Studio Code để có thể biên dịch được mã nguồn Python.

III. Chạy thử chương trình đầu tiên

Trong cuốn sách này, các chương trình nguồn được viết trên trình biên tập Sublime Text.

Mở Sublime Text lên và gõ lại đoạn code sau, lưu lại dưới tên "Chaoban.py":



Sau khi lưu lại bấm **"Ctrl + B"** đối với máy sử dụng hệ điều hành Windows hoặc Linux, và bấm **"Comand + B"** đối với máy sử dụng hệ điều hành Mac OS.

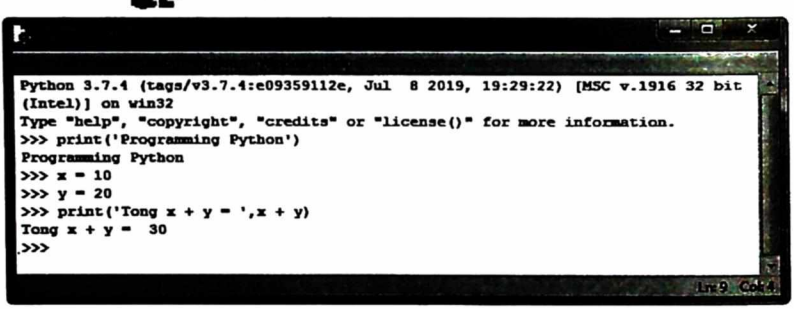
➢ **Chạy chương trình từ dòng lệnh**

Chương trình Python có thể thực hiện từ dòng lệnh như sau:

- Mở cửa sổ dòng lệnh bằng cách chạy chương trình cmd.exe
- Sau đó nhập lệnh python [đường dẫn]\Chaoban.py và bấm Enter.

➢ **Viết chương trình từ Python Shell**

Để soạn thảo và chạy chương trình Python ta cũng có thể sử dụng môi trường **IDLE (Python 3.7 32-bit)**



❖ ***Chú ý***

Một chương trình nguồn viết bằng Python có phần mở rộng là ***.py****. Khi thực hiện các chương trình này thì chương trình biên tập nhận diện được đây là một chương trình Python. Sau đó sử dụng trình "thông dịch Python" (Python Interpreter) để chạy chương trình.*

Chương 2. LẬP TRÌNH PYTHON CƠ BẢN

CHỦ ĐỀ 3. VÀO/RA DỮ LIỆU TỪ THIẾT BỊ CHUẨN

A. KIẾN THỨC CƠ BẢN

Thiết bị chuẩn dùng để vào/ra dữ liệu trong máy tính là bàn phím và màn hình. Trong bài này, ta sẽ đề cập tới các câu lệnh cho phép đưa dữ liệu vào từ bàn phím và đưa kết quả ra màn hình.

1. Đưa kết quả ra màn hình

1.1. Câu lệnh

Câu lệnh: `print(`<danh sách kết quả>`)`

☞ Dùng để đưa các kết quả ra màn hình.

1.2. Một số ví dụ

Ví dụ 1. Chương trình:

```
print("*")
print("**")
print("***")
print("****")
print("*****")
```

☞ Kết quả:

```
*
**
***
****
*****
```

Ví dụ 2. Chương trình:

```
print("*\n**\n***\n****\n*****")
```

☞ Kết quả:

```
*
**
***
****
*****
```

❖ ***Chú ý***

Kí tự xuống dòng và về đầu dòng trong Python là '\n';

Ví dụ 3. Chương trình:

```
x = "Chao mung ban den voi Python"
print(x)
```

☞ Kết quả:

```
Chao mung ban den voi Python
```

Ví dụ 4. Chương trình:

```
print("Ngon ngu","Lap trinh","Python")
```

☞ Kết quả:

```
Ngon ngu Lap trinh Python
```

❖ ***Nhận xét***

Khi đưa nhiều kết quả trong câu lệnh **`print()`**, ta sử dụng dấu phẩy để phân cách giữa các kết quả.

2. Câu lệnh `print()` với các tham số

Tham số `sep`

Câu lệnh: **`print`***(<kết quả 1>, <kết quả 2>, .., <kết quả n>)*

☞ Đưa *n* kết quả ra màn hình. Các kết quả sẽ được đưa ra trên một dòng và giữa các kết quả là một khoảng trắng (white space).

Ví dụ 5. Chương trình:

```
print("a","b","c")
```

☞ Kết quả:

```
a b c
```

Đây là sự mặc định của tham số **sep**, nhưng ta có thể thay đổi điều này bằng thiết lập tham số **sep** này theo câu lệnh:

print(*<kết quả 1>, .., <kết quả n>, **sep** = [nội dung giữa các kết quả]*)

Như vậy tham số ***sep*** mặc định đó là sep = "".

Ví dụ 6. Chương trình:

```
print("a","b","c", sep = "")
print("a","b","c", sep = "--")
print("a","b","c", sep = " ")
print("a","b","c", sep = ",")
```

☞ Kết quả:

```
abc
a--b--c
a b c
a,b,c
```

Giải thích:

Câu lệnh 1: **sep** = "", tức là rỗng, không có khoảng cách giữa các kết quả.

Câu lệnh 2: **sep** = "--", do vậy sẽ có -- giữa các kết quả.

Câu lệnh 3: **sep** = " ", tức là kí tự trắng, nên giữa các kết quả có kí tự trắng.

Câu lệnh 4: **sep** = ",", do vậy sẽ có dấu "," giữa các kết quả.

Tham số end

Câu lệnh: **print**(<kết quả>)

Khi thực hiện xong sẽ tự động xuống dòng và về đầu dòng mới. Muốn in nhiều kết quả trên cùng một dòng, ta có thể sử dụng tham số **end**.

print(*<kết quả>*, **end** = *[nội dung cuối mỗi kết quả]*)

Ví dụ 7. Chương trình:

```
print("a",end = " ")
print("b",end = "+")
print("c",end = "\n")
print("Tu hoc")
print("Python")
```

☞ Kết quả:

```
a b+c
Tu hoc
Python
```

Như vậy, tham số **end** mặc định bằng **end** = '\n', kí tự xuống dòng và về đầu dòng mới.

3. Nhập dữ liệu từ bàn phím

Câu lệnh: **input()**

☞ Dùng để nhập một dãy kí tự từ bàn phím.

☞ ***Một số ví dụ***

Ví dụ 8. Nhập một dãy kí tự và in ra màn hình.

Chương trình:

```
print("Moi nhap mot day ki tu")
x = input()
print("Day ki tu duoc nhap ",x)
```

☞ Kết quả chạy từ dòng lệnh **cmd.exe**

```
C:\Users\Administrator> python c:\python\nhap.py
Moi nhap mot day ki tu
Hoc lap trinh python
Day ki tu duoc nhap  Hoc lap trinh python
```

Ta có cách viết khác của câu lệnh **input()**

```
x = input("Moi nhap mot day ki tu")
print("Day ki tu duoc nhap ",x)
```

Ví dụ 9. Nhập hai số tự nhiên a và b. Đưa ra tổng $a + b$.

Chương trình:

```
a = int(input("Nhap so a: "))
b = int(input("Nhap so b: "))
print("Tong a + b: ",a+b)
```

❖ ***Chú ý***

Câu lệnh **input()** *trả về một chuỗi kí tự, vì vậy khi muốn kết quả trả về là các số nguyên ta cần ép kiểu như trên. Ví dụ* **int**("123") *sẽ chuyển chuỗi kí tự "123" thành số 123.*

Để chạy hai ví dụ trên, ta cần thực hiện chương trình ở chế độ dòng lệnh.

4. Chú thích

Khi viết chương trình, để làm rõ ý nghĩa thực hiện của một số câu lệnh nào đó, ta có thể sử dụng chú thích. Khi thực hiện chương trình, trình thông dịch sẽ "bỏ qua" các dòng chú thích này.

Cách viết chú thích:

a) Chú thích trên một dòng

 # nội dung cần chú thích

Ví dụ 10. Chương trình:

```
x = "Tu hoc Python"
# Dua bien x ra man hinh
print(x)
```

☞ Kết quả:

```
Tu hoc Python
```

b) Chú thích trên nhiều dòng

Để viết chú thích trên nhiều dòng, ta có thể sử dụng nhiều lần chú thích một dòng hoặc thực hiện như sau:

***Cách* 1**. Bắt đầu bằng 3 nháy đôi """ và kết thúc bằng 3 nháy đôi """.

“““

Nội dung cần chú thích

”””

Cách 2. Bắt đầu bằng 3 nháy đơn ‘‘‘ và kết thúc bằng 3 nháy đơn ’’’.

‘‘‘

Nội dung cần chú thích

’’’

Ví dụ 11. Chương trình:

```
"""
Python duoc xem la mot ngon ngu lap trinh phu
hop voi cac du an lon ve tri tue nhan tao.
Cac cau lenh cua Python trong sang va rat de su
dung.
"""
x = "Tu hoc Python"
# Dua bien x ra man hinh
print(x)
```

☛ Kết quả:

```
Tu hoc Python
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1

Dùng lệnh `print()` *để in ra màn hình các hình vẽ sau:*

a) *Hình chữ nhật*

```
**********
**********
**********
**********
```

b) *Hình cây thông*

```
*
***
*****
*******
*********
***********
*
*
*
```

Bài tập 2

Sử dụng hàm `input()` *nhập từ bàn phím hai số tự nhiên* a *và* b. *Đưa ra hiệu* $a - b$.

Bài tập 3

Hãy phân tích kết quả khi chạy chương trình dưới đây, sau đó chạy chương trình để đối chiếu kết quả.

```
a = 10
b = 20
c = 30
print(a,b,c,sep = "+", end = "=")
print(a+b+c)
```

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1

a) Hình chữ nhật.

```
print("************")
print("************")
print("************")
print("************")
```

b) Hình cây thông.

```
print("      *      ")
print("     ***     ")
print("    *****    ")
print("   *******   ")
print("  *********  ")
print("***********")
print("      *      ")
print("      *      ")
print("      *      ")
```

Bài tập 2

Chương trình:

```
a = int(input("Nhap so tu nhien a: "))
b = int(input("Nhap so tu nhien b: "))
c = a + b
print("Hieu a - b = ", c)
```

Bài tập 3

Kết quả khi chạy chương trình:

```
10+20+30=60
```

CHỦ ĐỀ 4. BIẾN VÀ GÁN GIÁ TRỊ CHO BIẾN

A. KIẾN THỨC CƠ BẢN

1. Khái niệm về biến

Biến (variable) là một đối tượng mà giá trị của nó có thể thay đổi được khi thực hiện chương trình. Mỗi biến đều được đặt tên và cần một lượng ô nhớ máy tính tương ứng để lưu giá trị của nó. Trong Python, mỗi biến là một con trỏ chỉ đến ô nhớ chứa giá trị đã được gán cho biến đó.

Khác với các ngôn ngữ lập trình khác, Python không có lệnh khai báo biến mà có lệnh gán trực tiếp giá trị cho biến theo cách:

<Tên biến> = <Giá trị>

Các ô nhớ trong máy tính được đánh địa chỉ (số thứ tự) bắt đầu từ 0. Để lấy địa chỉ ô nhớ lưu trữ giá trị của một biến, ta dùng hàm `id` (*Tên biến*).

2. Một số ví dụ về sử dụng biến

Ví dụ 1. Chương trình:

```
# Gan gia tri cho bien x
x = "Tu hoc Python"
print(x)
print("Dia chi o nho: ", id(x))
```

☞ Kết quả:

```
Tu hoc Python
Dia chi o nho:  7940120
```

❖ ***Chú ý***

Cùng một chương trình như trên, mỗi lần thực hiện, địa chỉ ô nhớ của biến x có thể khác nhau.

Ví dụ 2. Chương trình:

```
x = "Tu hoc Python"
print("Gia tri bien x: ",x)
print("Dia chi o nho:  ",id(x))
x = "Da thay doi gia tri"
print("Gia tri bien x: ",x)
print("Dia chi o nho:  ",id(x))
```

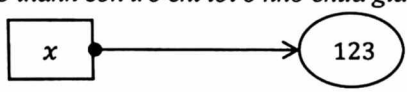
☞ Kết quả:

```
Gia tri bien x:  Tu hoc Python
Dia chi o nho:   19933168
Gia tri bien x:  Da thay doi gia tri
Dia chi o nho:   20110560
```

❖ ***Nhận xét***

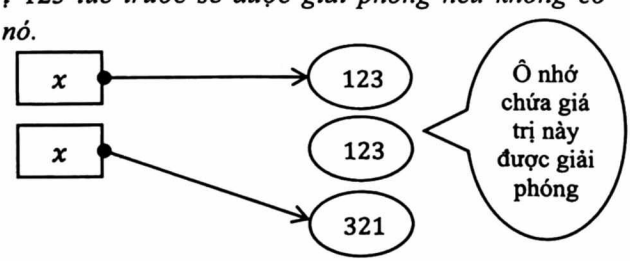
Trong chương trình trên, ta nhận thấy rằng, cùng một biến x nhưng có hai địa chỉ khác nhau. Điều này được lý giải do cơ chế cấp phát bộ nhớ cho biến trong Python như sau:

Chẳng hạn, khi gán x = 123, máy tính sẽ sử dụng một số ô nhớ (ở một vị trí nào đó trong bộ nhớ RAM – Random Access Memory*) để lưu giá trị 123 và biến x trở thành con trỏ chỉ tới ô nhớ chứa giá trị 123.*



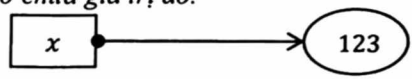
Khi ta gán x = 321*, máy tính sẽ dùng một số ô nhớ khác để chứa giá trị* 321*, và lúc này x sẽ chỉ tới ô nhớ chứa giá trị* 321*, ô nhớ chứa giá trị* 123 *lúc trước sẽ được giải phóng nếu không có biến nào chỉ tới nó.*

- x = 123
- x = 321



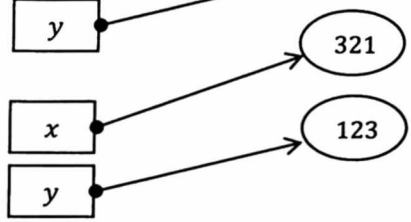
Trong trường hợp có nhiều biến có cùng giá trị, thì các biến này cùng chỉ tới một ô nhớ chứa giá trị đó.

- x = 123



- x = 123
  y = 123

- x = 321
  y = 123



Ví dụ 3. Chương trình:

```
x = 123
y = 123
print("Dia chi o nho x tro toi: ",id(x))
print("Dia chi o nho y tro toi: ",id(y))
x = 321
print("Dia chi o nho x tro toi: ",id(x))
print("Dia chi o nho y tro toi: ",id(y))
```

☞ Kết quả:

```
Dia chi o nho x tro toi:  1457220608
Dia chi o nho y tro toi:  1457220608
Dia chi o nho x tro toi:  3267760
Dia chi o nho y tro toi:  1457220608
```

❖ ***Chú ý***

Các ô nhớ được sử dụng để chứa giá trị của các biến, trong quá trình thay đổi giá trị, nếu ô nhớ nào không có biến chỉ tới thì ô nhớ đó sẽ được giải phóng để chứa các giá trị khác trong quá trình thực hiện chương trình.

3. Một số quy tắc về đặt tên biến

Việc đặt tên biến được quy định bởi từng ngôn ngữ lập trình và hệ điều hành tương ứng. Việc đặt tên biến trong Python được quy định như sau:

- *Biến có thể chứa các kí tự chữ cái và chữ số.*

- *Không chứa kí tự trống "space" và các kí tự đặc biệt khác như* +, −, *, ..

- *Tên biến không thể bắt đầu bằng kí tự số.*

- *Không sử dụng các từ khóa, tên hàm đã được ngôn ngữ lập trình sử dụng.*

- *Tên biến nên đặt ngắn gọn, gợi nhớ và dễ nhớ.*

Ví dụ 4

- Tên đúng: x, y, x1, x2.
- Tên sai: 1x, 2x, a b, de f.

4. Một số cách rút gọn để gán giá trị cho biến

Python cho phép người lập trình viết ngắn gọn các biểu thức gán giá trị, điều này đã tạo ra các câu lệnh rất đẹp và thanh lịch.

Cách gán thông thường	Cách gán rút gọn
a = 2 b = 2 c = 2	a = b = c = 2
a = 1 b = 2 c = 3	a, b, c = 1, 2, 3
Tráo đổi giá trị a và b x = a a = b b = x	a, b = b, a

B. BÀI TẬP ÔN LUYỆN

Bài tập 1

Chương trình nào dưới đây là chương trình đúng, chương trình sai? Hãy giải thích những chỗ sai.

Chương trình 1:

```
a = b = c
print(a)
print(b)
print(c)
```

Chương trình 2:

```
a = 1
a = b = c
print(a)
print(b)
print(c)
```

Chương trình 3:

```
c = 1
a = b = c
print(a)
print(b)
print(c)
```

Bài tập 2

Không chạy chương trình, hãy đưa ra kết quả có được khi chương trình được thực hiện?

```
x = 2020
print(x)
print("x")
```

Bài tập 3

Chương trình sau có lỗi không, tại sao?

```
x = 2020
2020 = x
print(x)
```

Bài tập 4. Không chạy chương trình, hãy đưa ra kết quả có được khi chương trình được thực hiện?

```
a, b, c = 1, 2, 3
a, b, c = b, c, a
print(a,b,c)
```

Bài tập 5. Không chạy chương trình, hãy đưa ra kết quả có được khi chương trình được thực hiện?

```
a = 1
b = 2
a, b = b, a + b
print(a, b)
```

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1

Chương trình 1:

Có lỗi ở dòng 1: a = b = c. Bởi vì, theo trình tự thực hiện, các biến a, b sẽ nhận giá trị của biến c. Nhưng biến c là lần đầu tiên xuất hiện trong chương trình và nó chưa được gán giá trị. Điều này dẫn đến biến không xác định.

Chương trình 2:

Ở dòng 1, biến b được gán giá trị 1, nhưng ở dòng 2: a = b = c là câu lệnh sai. Vì lúc này biến a, b nhận giá trị biến c, nhưng biến c lần đầu tiên xuất hiện trong chương trình và nó chưa được gán giá trị.

Chương trình 3:

Hoàn toàn đúng vì đã khắc phục được sự không xác định của biến c bằng cách gán c = 1.

Bài tập 2

Ở dòng 1, biến x được gán giá trị bằng 2020, dòng 2 là câu lệnh `print`(x) đưa giá trị x ra màn hình, dòng 3 câu lệnh `print`("x") đưa kí tự x ra màn hình. Vì vậy chương trình sẽ có kết quả:

```
2020
x
```

Bài tập 3

Chương trình có lỗi ở dòng 2, vì 2020 không phải là biến nên không gán giá trị bằng x được.

Bài tập 4

Ở dòng 1, khi thực hiện câu lệnh $a, b, c = 1, 2, 3$, ta có $a = 1$, $b = 2, c = 3$.

Ở dòng 2, khi thực hiện câu lệnh $a, b, c = b, c, a$ thì $a \leftarrow b$; $b \leftarrow c$ và $c \leftarrow a$ (giá trị cũ). Tức là $a = 2, b = 3, c = 1$.

Do vậy, kết quả khi thực hiện **print**(a, b, c) là 2 3 1.

Bài tập 5

Ở dòng 3, khi thực hiện câu lệnh $a, b = b, a + b$ thì $a \leftarrow b$; $b \leftarrow a + b$ (giá trị cũ). Tức là $a = 2, b = 3$.

Do vậy, kết quả khi thực hiện `print(a, b)` là 2 3.

CHỦ ĐỀ 5. KIỂU DỮ LIỆU SỐ

A. KIẾN THỨC CƠ BẢN

Ngôn ngữ lập trình Python có hỗ trợ 3 kiểu dữ liệu về số:

- Số nguyên – int
- Số thực – float
- Số phức - complex

1. Kiểu số nguyên - int

Ví dụ 1. Chương trình:

```
x = 12
y = 13
print("Tong hai so bang: ",x + y)
print("Tich hai so bang: ",x*y)
```

☞ Kết quả:

```
Tong hai so bang: 25
Tich hai so bang: 156
```

Các phép toán với số nguyên

Phép toán	Kí hiệu	Ví dụ
Cộng	+	5 + 2 = 7
Trừ	−	5 − 2 = 3
Nhân	*	5 * 2 = 10
Lũy thừa	**	5 ** 2 = 25
Chia	/	5/2 = 2.5
Chia lấy thương	//	5//2 = 2
Chia lấy dư	%	5%2 = 1

❖ ***Chú ý***

Python cho phép tính toán với các số nguyên có giá trị rất lớn.

Ví dụ 2. Chương trình:

```
a = 2020
n = 50
s = a**n
print(a," Luy thua ",n," bang:\n ",s)
```

☞ Kết quả:

```
2020  Luy thua  50  bang:
185169081500444162204657163114373271945042602530683
433245909302930565529940786801367978629508418437200
129280666396262400000000000000000000000000000000000
0000000000000000
```

2. Kiểu số thực – float

Ví dụ 3. Chương trình:

```
pi = 3.14
r  = 2
s  = pi*r*r
print("Dien tich: ",s)
```

☞ Kết quả:

```
Dien tich: 12.56
```

Các phép toán trên số thập phân:

Phép toán	**Kí hiệu**	**Ví dụ**
Cộng	+	5.0 + 2.0 = 7.0
Trừ	−	5.0 − 2.0 = 3.0
Nhân	*	5.0 * 2.0 = 10.0
Lũy thừa	**	25 ** 0.5 = 5.0
Chia	/	5/2 = 2.5

❖ ***Chú ý* 1**

Python cho phép viết các số thực dưới dạng kí pháp khoa học của nó, chẳng hạn, $3.0 * 10^3$ *có thể viết* 3.0e3, $3.0 * 10^{-3}$ *có thể viết* $3.0e-3$.

Ví dụ 4. Chương trình:

```
a = 2e10
b = 2e-10
c = a*b
print("Gia tri cua a: ", a)
print("Gia tri cua c: ", c)
```

☞ Kết quả:

```
Gia tri cua a:  20000000000.0
Gia tri cua c:  4.0
```

3. Số phức – complex

Một số phức có dạng $a + bj$. Trong đó a, b là các số thực; a được gọi là phần thực, b được gọi là phần ảo.

Ví dụ 5. Chương trình:

```
x = 2 + 1j
y = 3 - 2j
print("Tong bang: ", x + y)
print("Hieu bang: ", x - y)
print("Tich bang: ", x * y)
```

☞ Kết quả:

```
Tong bang:  (5-1j)
Hieu bang:  (-1+3j)
Tich bang:  (8-1j)
```

Ví dụ 6. Chương trình:

```
x = complex(1, 2)
y = complex(2)
print("x = ", x)
print("y = ", y)
```

☞ Kết quả:

```
x =  (1+2j)
y =  (2+0j)
```

Ví dụ 7. Chương trình:

```
x = 2
y = 2.0
z = 2 + 1j
print("Kieu cua bien x: ",type(x))
print("Kieu cua bien y: ",type(y))
print("Kieu cua bien z: ",type(z))
```

☞ Kết quả:

```
Kieu cua bien x:  <class 'int'>
Kieu cua bien y:  <class 'float'>
Kieu cua bien z:  <class 'complex'>
```

❖ ***Nhận xét***

- *Hàm lấy kiểu dữ liệu của biến*: **type**(<tên biến>)
- *Hàm tạo số phức*: **complex**(a, b) = $a + bj$; **complex**(a) = $a + 0j$.

4. Hàm tạo số ngẫu nhiên

Python cung cấp mô-đun **random** cho phép sinh các số ngẫu nhiên cả về số nguyên và số thực.

- Hàm **randint**(a, b) tạo ngẫu nhiên một số nguyên thuộc $[a,b]$.
- Hàm **uniform**(a, b) tạo ngẫu nhiên một số thực $[a,b]$.

Để sử dụng hai câu lệnh này, cần phải import chúng từ mô-đun **random** bằng câu lệnh:

```
from random import randint, uniform
```

Ví dụ 8. Chương trình:

```
from random import randint, uniform
x = randint(-10, 10)
y = uniform(-100, 100)
print("So nguyen tao duoc: ", x)
print("So thuc tao duoc: ", y)
```

☞ Kết quả:

```
So nguyen tao duoc:  -6
So thuc tao duoc:  28.20908252310869
```

5. Các hàm toán học thông dụng khác

Ngôn ngữ lập trình Python cung cấp đầy đủ các hàm toán học thông dụng:

Tên hàm	Thực hiện
sin(x)	Tính sin của một góc có số đo x (rad)
cos(x)	Tính côsin của một góc có số đo x (rad)
tan(x)	Tính tang của một góc có số đo x (rad)
sqrt(x)	Tính căn bậc hai của x
trunc(x)	Tính phần nguyên của x
pi	Tính giá trị của số pi
round(x, n)	Làm tròn x đến n chữ số thập phân
abs(x)	Tính giá trị tuyệt đối x
max($a_1, a_2, .., a_n$)	Trả về giá trị lớn nhất trong các số hạng $a_1, a_2, .., a_n$
min($a_1, a_2, .., a_n$)	Trả về giá trị nhỏ nhất trong các số hạng $a_1, a_2, .., a_n$

❖ ***Chú ý 2***

Để sử dụng các hàm toán học trong Python (trừ các hàm **abs()**, **min()**, **max()** *và* **round()** *), ta cần sử dụng câu lệnh*: **from math import**

Ví dụ 9. Chương trình:

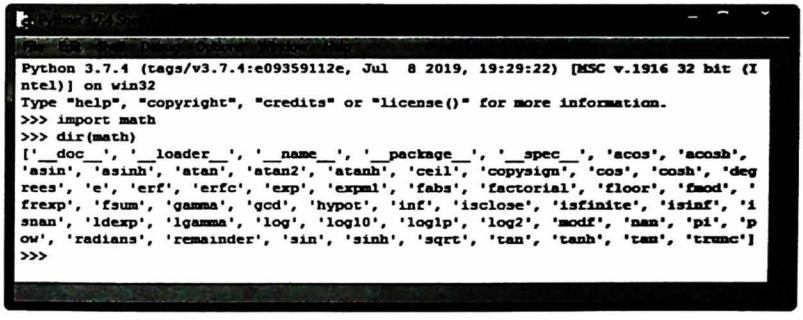
```
from math import pi, trunc
print("Gia tri cua Pi:          ",pi)
print("Phan nguyen cua Pi:      ",trunc(pi))
print("Lam tron Pi den 2 chu so: ",round(pi,2))
```

☞ Kết quả:

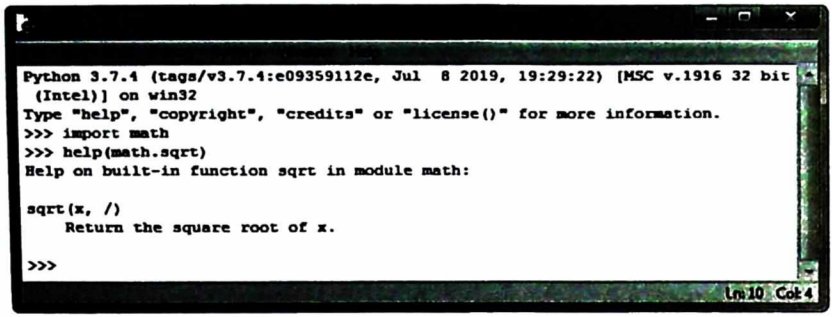
```
Gia tri cua Pi:            3.141592653589793
Phan nguyen cua Pi:        3
Lam tron Pi den 2 chu so:  3.14
```

❖ *Chú ý 3*

- *Để* **import** *tất cả các hàm trong mô-đun* **math**, *ta thực hiện*: **from math import ***
- *Python cho phép tra cứu các hàm toán học cũng như trợ giúp việc sử dụng các hàm đó. Để tra cứu bạn mở* Python 3.7.4 Shell *và thực hiện*:



- *Để tìm hiểu ý nghĩa thực hiện của một hàm nào trong thư viện* **math**, *ta có thể thực hiện*:



6. Một số cách viết phép toán rút gọn

Biểu thức	Rút gọn
n = n + a	n += a
n = n - a	n -= a
n = n * a	n *= a
n = n / a	n /= a
n = n // a	n //= a
n = n ** a	n **= a

7. Những lưu ý về sai số

Khi thực hiện tính toán với các số thực, máy tính có thể dẫn đến những sai lệch về giá trị.

Ví dụ 10. chương trình:

```
a = 1/3
print(a)
```

☞ Kết quả:

```
0.3333333333333333
```

Ví dụ 11. chương trình:

```
a = 1/3
print(a == 0.3333333333333333)
```

```
print(a == 0.333333333333333)
print(a == 0.3333333333333333)
print(a == 0.33333333333333333335678)
```

☞ Kết quả:

```
True
False
True
True
```

Ví dụ 12. chương trình:

```
a = .3              # .3 == 0.3
b = .6              # .6 == 0.6
c = 0.9
print(a+b)
print(a+b == c)
```

☞ Kết quả:

```
0.8999999999999999
False
```

Ví dụ 13. chương trình:

```
a = .3              # .3 == 0.3
b = .6              # .6 == 0.6
c = 0.9
print(round(a+b,2) == round(c,2))
```

☞ Kết quả:

```
True
```

❖ **Nhận xét**

Khi tính toán với các giá trị là số thực, ta nên sử dụng hàm làm tròn `round()` để có thể quản lý những kết quả nhận được.

B. BÀI TẬP ÔN LUYỆN

Viết chương trình thực hiện công việc sau:

Bài tập 1

Nhập số nguyên dương a có 3 chữ số. Tính tổng các chữ số của a.

Bài tập 2

Nhập ba số nguyên a, b, c. Đưa ra tổng giá trị của số hạng nhỏ nhất và số hạng lớn nhất, tức là $min(a, b, c) + max(a, b, c)$.

Bài tập 3

Cho đường tròn có bán kính R. Tính chu vi của đường tròn đó, làm tròn đến 2 chữ số thập phân.

Bài tập 4

Nhập ba số nguyên dương a, b, c. Hãy tìm ước chung lớn nhất của 3 số a, b, c.

Bài tập 5

Nhập hai số nguyên dương a, b. Tìm bội chung nhỏ nhất của a và b.

C. THUẬT TOÁN HƯỚNG DẪN GIẢI

Bài tập 1

Ý tưởng thuật toán

Để tính tổng các chữ số, ta sẽ tách từng chữ số của a rồi cộng lại.

- Chữ số hàng đơn vị bằng $a\%10$, hai chữ số còn lại tạo thành số x bằng $a//10$.

- Chữ số hàng chục của a bằng $x\%10$.

- Chữ số hàng trăm của a bằng $x//10$.

Chương trình

```
a = int(input("Nhap so tu nhien co 3 chu so: "))
donvi = a%10
x = a//10
hangchuc = x%10
hangtram = x//10
tongchuso = donvi + hangchuc + hangtram
print("Tong chu so: ",tongchuso)
```

Bài tập 2

Ý tưởng thuật toán

Sử dụng hàm $min(a, b, c)$ và hàm $max(a, b, c)$ để tìm giá trị nhỏ nhất và lớn nhất của ba số a, b, c, sau đó cộng hai giá trị vừa tính được.

Chương trình

```
a = int(input("Nhap so nguyen a: "))
b = int(input("Nhap so nguyen b: "))
c = int(input("Nhap so nguyen c: "))
min_abc = min(a,b,c)
max_abc = max(a,b,c)
tong_min_max = min_abc + max_abc
print("Tong min max: ",tong_min_max)
```

Bài tập 3

Ý tưởng thuật toán

Để tính chu vi đường tròn, ta sử dụng công thức tính chu vi bằng $2\pi R$. Giá trị π được lấy trong mô-đun `math` và sử dụng hàm `round()` để làm tròn đến hai chữ số thập phân.

Chương trình

```
from math import pi
R = 2020
chuvi = round(2*pi*R,2)
print("Chu vi bang: ", chuvi)
```

Bài tập 4

Ý tưởng thuật toán

- Sử dụng hàm $\boldsymbol{gcd(a, b)}$ trong mô-đun $\boldsymbol{math}$ để tìm ước chung lớn nhất của hai số a và b.

- Ta đặt $\boldsymbol{d = gcd(a, b)}$, khi đó, ước chung lớn nhất của 3 số a, b, c sẽ bằng $gcd(d, c)$.

Chương trình

```
from math import gcd
a = int(input("Nhap a = "))
b = int(input("Nhap b = "))
c = int(input("Nhap c = "))
d = gcd(a, b)
print("Uoc chung lon nhat cua a, b, c: ",gcd(d, c))
```

Bài tập 5

Ý tưởng thuật toán

- Ta có, bội chung nhỏ nhất của hai số nguyên dương a và $b = a*b//\ gcd(a,b)$.

Chương trình

```
from math import gcd
a = int(input("Nhap a = "))
b = int(input("Nhap b = "))
print("Boi chung nho nhat a, b: ", a*b//gcd(a, b))
```

CHỦ ĐỀ 6. CẤU TRÚC RẼ NHÁNH

A. KIẾN THỨC CƠ BẢN

1. Câu lệnh `if`:

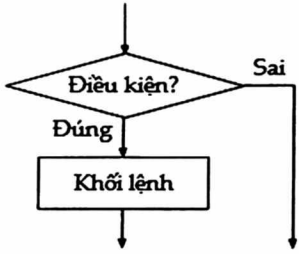
Cú pháp:

```
if <Điều kiện>:
    <Khối lệnh>
```

Thực hiện:

Nếu <Điều kiện> đúng thì sẽ thực hiện <Khối lệnh>.

Sơ đồ thực hiện:



Ví dụ 1. Chương trình:

```
a = 10
b = 20
if a < b:
    print("a nho hon b")
```

☞ Kết quả:

```
a nho hon b
```

Ví dụ 2. Chương trình:

```
a = 10
b = 20
if a < b:
    print("a nho hon b")
    print("Tong a va b: ", a + b)
```

☞ Kết quả:

```
a nho hon b
Tong a va b:  30
```

Ví dụ 3. Chương trình:

```
a = 30
b = 20
if a < b:
    print("a nho hon b")
    print("Tong a va b: ", a + b)
print("Vi tri ngoai khoi lenh")
```

☞ Kết quả:

```
Vi tri ngoai khoi lenh
```

❖ **Khái niệm khối lệnh**

Khối lệnh trong Python được định nghĩa gồm một lệnh hoặc nhiều lệnh liên tiếp được viết thẳng cột với nhau. Các câu lệnh này được gọi là cùng mức (level). Như vậy, có thể hiểu, khối lệnh là một dãy lệnh liên tiếp cùng mức.

```
if a < b:
    print("a nho hon b")
    print("Tong a va b: ", a + b)
```

} Khối lệnh

❖ **Chú ý**: Ta có thể viết gọn trên một hàng cho câu lệnh **if** như sau:

```
if a < b: print("a nho hon b")
```

2. Câu lệnh if:

else:

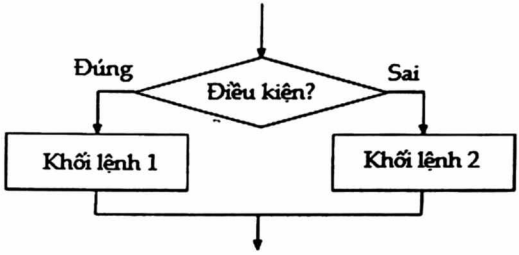
Cú pháp:

```
if <Điều kiện>:
    <Khối lệnh 1>
else:
    <Khối lệnh 2>
```

Thực hiện:

Nếu <Điều kiện> đúng thì thực hiện <Khối lệnh 1>; ngược lại (*điều kiện sai*) thực hiện <Khối lệnh 2>.

Sơ đồ thực hiện



Ví dụ 4. Chương trình:

```
a = 20
b = 20
if a == b:
    print("a va b bang nhau")
else:
    print("a va b khac nhau")
```

☞ Kết quả:

```
a va b bang nhau
```

Ví dụ 5. Chương trình:

```
a = 2
b = 20
if a == b:
    print("a va b bang nhau")
else:
    print("a va b khac nhau")
```

☞ Kết quả:

```
a va b khac nhau
```

❖ *Chú ý*

`if` và `else` cần phải viết thẳng cột (cùng một mức).

3. Câu lệnh `if`:

`elif`:

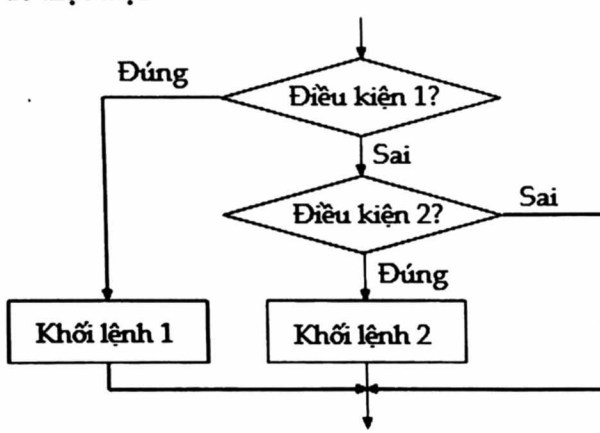
Cú pháp

```
if    <Điều kiện 1>:
          <Khối lệnh 1>
elif <Điều kiện 2>:
          <Khối lệnh 2>
```

Thực hiện:

Nếu <Điều kiện 1> đúng sẽ thực hiện <Khối lệnh 1>, ***ngược lại***, nếu <Điều kiện 2> đúng sẽ thực hiện <Khối lệnh 2>.

Sơ đồ thực hiện



Ví dụ 6. Chương trình:

```
a = 20
b = 30
if a == b:
   print("a va b bang nhau")
```

```
elif a < b:
    print("a nho hon b")
```

☞ Kết quả:

```
a nho hon b
```

Ví dụ 7. Chương trình:

```
a = 200
b = 30
if a == b:
    print("a va b bang nhau")
elif a < b:
    print("a nho hon b")
else:
    print("a lon hon b")
```

☞ Kết quả:

```
a lon hon b
```

4. Các phép toán so sánh và phép toán lôgic

Để viết các điều kiện trong cấu trúc rẽ nhánh, ta sử dụng các phép toán so sánh và phép toán lôgic.

a) Phép toán so sánh

Nội dung	Kí hiệu
So sánh bằng	==
So sánh khác	! =
So sánh nhỏ hơn	<
So sánh nhỏ hơn hoặc bằng	<=
So sánh lớn hơn	>
So sánh lớn hơn hoặc bằng	>=

Kết quả của phép so sánh là ***True*** hoặc ***False***.

Ví dụ 8. So sánh: 5 == 5 kết quả là ***True***; so sánh: 5 == 6 kết quả là ***False***.

b) Phép toán lôgic

Ngôn ngữ lập trình Python hỗ trợ các phép toán lôgic: **not**, **and**, **or**.

Bảng chân lý:

A	b	**not** a	a **and** b	a **or** b
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Ví dụ 9. Kiểm tra điều kiện ba số dương là ba cạnh của một tam giác.

Chương trình:

```
a = 10
b = 20
c = 15
if (a + b > c) and (b + c > a) and (c + a > b):
   print("a, b, c la ba canh cua mot tam giac")
else:
   print("a, b, c khong phai la ba canh cua mot tam giac")
```

☞ Kết quả:

```
a, b, c la ba canh cua mot tam giac
```

Ví dụ 10. Kiểm tra tính chẵn - lẻ của hai số.

Chương trình:

```
a = 10
b = 11
if (a%2 == 0) or (b%2 == 0):
    print("Trong hai so a, b co it nhat mot so
    chan")
else:
    print("Hai so a, b deu la so le")
```

☞ Kết quả:

```
Trong hai so a, b co it nhat mot so chan
```

❖ ***Chú ý***

Các phép toán lôgic có thứ tự ưu tiên thực hiện thấp hơn (ưu tiên sau) các phép toán số học và phép toán so sánh. Vì vậy, trong biểu thức điều kiện, ta có thể viết:

`if (a + b > c) and (b + c > a) and (c + a > b):`

Có thể viết: `if a + b > c and b + c > a and c + a > b:`

`if (a%2 == 0) or (b%2 == 0):`

Có thể viết: `if a%2 == 0 or b%2 == 0:`

5. Cách viết gọn của một số biểu thức điều kiện

Cách viết đầy đủ	Cách viết rút gọn
`if` <điều kiện> == True:	`if` <điều kiện>:
`if` a == 0 and b == 0 and c == 0:	`if` a == b == c:
`if` 1 < a and a < b and b < 5:	`if` 1 < a < b < 5:

B. BÀI TẬP ÔN LUYỆN

Sử dụng các cấu trúc rẽ nhánh để giải các bài toán sau.

Bài tập 1

So sánh kết quả

Không chạy chương trình, hãy so sánh kết quả của hai chương trình sau:

Chương trình 1

```
a = 200
b = 30
if(a < b):
   print("a nho hon b")
   print("Tong a + b = ", a+b)
print("b nho hon a")
```

Chương trình 2

```
a = 200
b = 30
if(a < b):
   print("a nho hon b")
print("Tong a + b = ", a+b)
print("b nho hon a")
```

Bài tập 2. Giải phương trình bậc 2

Nhập vào ba số a, b, c (a ≠0). Giải phương trình bậc hai $ax^2 + bx + c = 0$.

Bài tập 3. Phân loại tam giác

Nhập vào ba số a, b, c *nguyên dương. Kiểm tra xem* a, b, c *có phải ba cạnh của một tam giác hay không? Nếu là ba cạnh của một tam giác hãy phân loại xem tam giác có ba cạnh* a, b, c *đó thuộc loại nào dưới đây:*

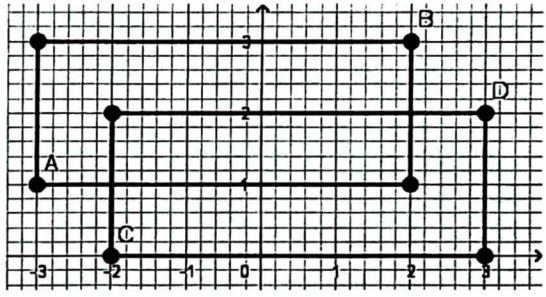
1. *Tam giác đều;*
2. *Tam giác vuông;*
3. *Tam giác thường.*

Bài tập 4. Giao đoạn

Nhập bốn số nguyên a, b, c, d ($a < b, c < d$). Tính xem có bao nhiêu số nguyên thuộc đoạn giao $[a; b] \cap [c; d]$, tức là có bao nhiêu số nguyên x thỏa mãn: $a \leq x \leq b$ và $c \leq x \leq d$. Ví dụ $a = 1; b = 6; c = 3; d = 5$ $[a; b] = [1; 6]$, $[c; d] = [3; 5], [1; 6] \cap [3; 5] = [3; 5]$. Như vậy có 3 số nguyên thuộc [3; 5].

Bài tập 5. Giao hai hình chữ nhật

Cho hai hình chữ nhật có cạnh song song hoặc trùng với các trục tọa độ. Hình chữ nhật thứ nhất được xác định bởi điểm trái dưới $A(x_A, y_A)$ và điểm phải trên $B(x_B, y_B)$. Hình chữ nhật thứ hai được xác định bởi điểm trái dưới $C(x_C; y_C)$, điểm phải trên $D(x_D, y_D)$. Xét xem hai hình chữ nhật có giao nhau hay không? Nếu giao nhau (có điểm trong chung) thì tính diện tích phần giao nhau.



Bài tập 6. Xếp loại học sinh

Cho điểm thi Toán, Lý, Hóa, Sinh, Tin của học sinh. Hãy xếp loại kết quả của học sinh theo tiêu chí:

Điểm trung bình (ĐTB)	Xếp loại
9 ≤ ĐTB	Xuất sắc
8 ≤ ĐTB < 9	Giỏi
7 ≤ ĐTB < 8	Khá
5 ≤ ĐTB < 7	Trung bình
ĐTB < 5	Yếu

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. So sánh kết quả

Chương trình 1: Ta nhận thấy khối lệnh trong câu lệnh `if:`

```
if a < b:
    print("a nho hon b")
    print("Tong a + b = ", a+b)
```

và với a = 200, b = 30 nên điều kiện không đúng. Do vậy các câu lệnh:

```
print("a nho hon b")
print("Tong a + b = ", a+b)
```

sẽ không được thực hiện. Chương trình sẽ thực hiện câu lệnh:

```
print("b nho hon a")
```

Chương trình 2: Khối lệnh trong câu lệnh `if:`

```
if a < b:
    print("a nho hon b")
```

chương trình cũng không thực hiện khối lệnh này. Chương trình sẽ thực hiện câu lệnh:

```
print("Tong a + b = ", a+b)
print("b nho hon a")
```

Như vậy, kết quả:

Chương trình 1	Chương trình 2
`b nho hon a`	`Tong a + b = 230` `b nho hon a`

Bài tập 2. Giải phương trình bậc 2

Ý tưởng thuật toán

Ta có các bước giải phương trình bậc 2 là:

Bước 1. Tính Delta $\Delta = b^2 - 4ac$;

Bước 2. Kiểm tra Δ âm hay dương?

Nếu $\Delta < 0$ thì phương trình vô nghiệm,

Nếu $\Delta = 0$ thì phương trình có nghiệm kép $x = \frac{-b}{2a}$,

Nếu $\Delta > 0$ thì phương trình có hai nghiệm phân biệt $x_1 = \frac{-b+\sqrt{\Delta}}{2a}$ và $x_2 = \frac{-b-\sqrt{\Delta}}{2a}$.

Chương trình

```
a = float(input("nhap he so a: "))
b = float(input("nhap he so b: "))
c = float(input("nhap he so c: "))
delta = b*b - 4*a*c
if delta < 0: print("Phuong trinh vo nghiem.")
elif delta == 0:
    x = -b/(2*a)
    print("Phuong trinh co nghiem kep bang: ",x)
else:
      from math import sqrt
      x1 = (-b + sqrt(delta))/(2*a)
      x2 = (-b - sqrt(delta))/(2*a)
      print("Phuong trinh co hai nghiem phan
biet:")
      print(x1)
      print(x2)
```

❖ ***Chú ý***

- $\sqrt{delta} = delta^{\frac{1}{2}} = delta$ ** (1/2).
- *Nếu tính căn bậc hai của* ***delta*** *theo hàm* **sqrt**(*delta*), *ta cần sử dụng câu lệnh* **from math import sqrt.**

Bài tập 3. Phân loại tam giác

Ý tưởng thuật toán

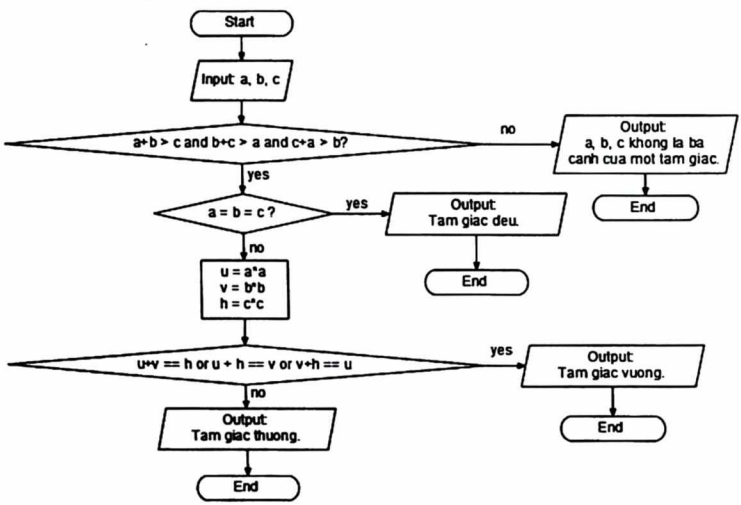
Ta có điều kiện để ba số dương a, b, c là độ dài của ba cạnh trong một tam giác là: $a + b > c$; $b + c > a$; $a + c > b$.

Khi a, b, c là ba cạnh trong một tam giác thì:

Nếu a = b = c thì có tam giác đều.

Nếu $a^2 = b^2 + c^2$ hoặc $b^2 = a^2 + c^2$ hoặc $c^2 = a^2 + b^2$ thì có tam giác đều, ngược lại thì có tam giác thường.

Sơ đồ thuật toán



Chương trình

```
a = int(input("nhap so nguyen duong a: "))
b = int(input("nhap so nguyen duong b: "))
c = int(input("nhap so nguyen duong c: "))
if a + b > c and b + c > a and c + a > b:
   # kiem tra tam giac deu
   if a == b == c:
        print("Tam giac deu.")
```

```
    elif a*a == b*b + c*c or b*b == a*a + c*c or
c*c==a*a+b*b:
        print("Tam giac vuong.")
    else:
        print("Tam giac thuong.")
else:
    print("a, b, c khong phai ba canh mot tam giac.")
```

Bài tập 4. Giao đoạn

Ý tưởng thuật toán

Ta nhận thấy một số $x \in [a;b] \leftrightarrow a \leq x \leq b$; $x \in [c;d] \leftrightarrow c \leq x \leq d$. Chính vì vậy $x \in [a;b] \cap [c;d] \leftrightarrow \begin{cases} a \leq x \leq b \\ c \leq x \leq d \end{cases} \leftrightarrow max(a,c) \leq x \leq min(b,d)$.

Như vậy, khi đặt $u = max(a,c)$, $v = min(b,d)$ thì có:

- Nếu $u > v$ thì rõ ràng không có số x nào thỏa mãn $u \leq x \leq v$, tức là không có số nguyên nào thuộc $[a;b] \cap [c;d]$.
- Nếu $u \leq v$ thì có $v - u + 1$ số x thuộc $[a;b] \cap [c;d]$.

Chương trình

```
a = int(input("nhap so nguyen a: "))
b = int(input("nhap so nguyen b: "))
c = int(input("nhap so nguyen c: "))
d = int(input("nhap so nguyen d: "))
u = max(a, c)
v = min(b, d)
if u > v:
    print("Khong co so nguyen nao thuoc giao [a;b] va [c;d]"
else:
    n = v - u + 1
    print("Co ",n," so nguyen thuoc giao [a;b] va [c;d]")
```

Bài tập 5. Giao hình chữ nhật

Ý tưởng thuật toán

Bài tập này là sự mở rộng từ bài tập 3.

Ta nhận thấy một điểm $M(x; y)$ thuộc hình chữ nhật có đỉnh trái dưới $A(x_A; y_A)$ và đỉnh phải trên $B(x_B; y_B)$ khi và chỉ khi: $\begin{cases} x_A \le x \le x_B \\ y_A \le y \le y_B \end{cases}$.

Tương tự như vậy, điểm M thuộc hình chữ nhật có đỉnh trái dưới $C(x_C; y_C)$ và đỉnh phải trên $D(x_D; y_D)$ khi và chỉ khi: $\begin{cases} x_C \le x \le x_D \\ y_C \le y \le y_D \end{cases}$.

Như vậy, để điểm M thuộc giao cả hai hình chữ nhật thì tọa độ $(x; y)$ thỏa mãn:

$$\begin{cases} x_A \le x \le x_B \\ x_C \le x \le x_D \\ y_A \le y \le y_B \\ y_C \le y \le y_D \end{cases} \leftrightarrow \begin{cases} max(x_A, x_C) \le x \le min(x_B, x_D) \\ max(y_A, y_C) \le y \le min(y_B, y_D) \end{cases} \quad (*)$$

Ta đặt: $xmin = max(x_A, x_C), xmax = min(x_B, x_D)$

$$ymin = max(y_A, y_C), ymax = min(y_B, y_D)$$

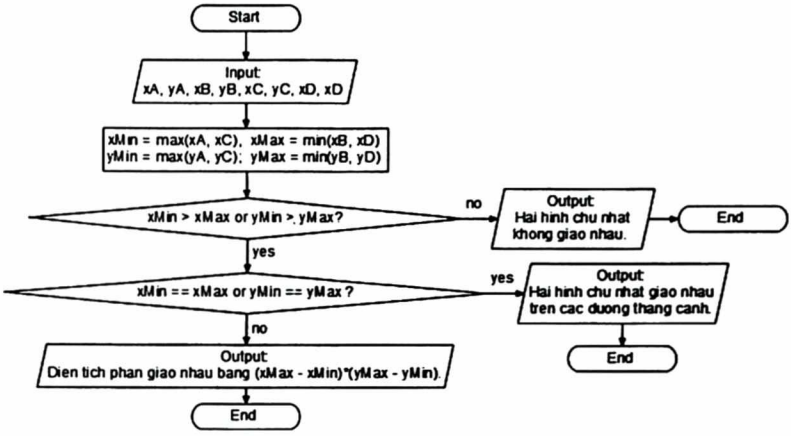
Thì có:

- Nếu $xmin > xmax$ hoặc $ymin > ymax$ thì hai hình chữ nhật này không giao nhau vì không có giá trị x hoặc y thỏa mãn điều kiện (*).

- Nếu $xmin = xmax$ hoặc $ymin = ymax$ thì hai hình chữ nhật sẽ tiếp xúc nhau trên các cạnh.

- Nếu $xmin < xmax$ và $ymin < ymax$ thì hai hình chữ nhật sẽ giao nhau bởi một hình chữ nhật có một cạnh là $xmax - xmin$ và một cạnh là $ymax - ymin$. Do vậy diện tích của hình chữ nhật giao là $(xmax - xmin) \times (ymax - ymin)$.

Sơ đồ thuật toán



Chương trình

```
print("nhap toa do diem A:")
xa = int(input())
ya = int(input())
print("nhap toa do diem B:")
xb = int(input())
yb = int(input())
print("nhap toa do diem C:")
xc = int(input())
yc = int(input())
print("nhap toa do diem D:")
xd = int(input())
yd = int(input())
xmin = max(xa, xc)
xmax = min(xb, xd)
ymin = max(ya, yc)
ymax = min(yb, yd)
```

```
    if xmin > xmax or ymin > ymax:
        print("Hai hinh chu nhat khong giao nhau: ")
    elif xmin == xmax or ymin == ymax:
        print("Hai hinh chu nhat giao nhau tren cac
canh: ")
    else:
        u = xmax - xmin
        v = ymax - ymin
        print("Dien tich hinh chu nhat giao bang:
",u*v)
```

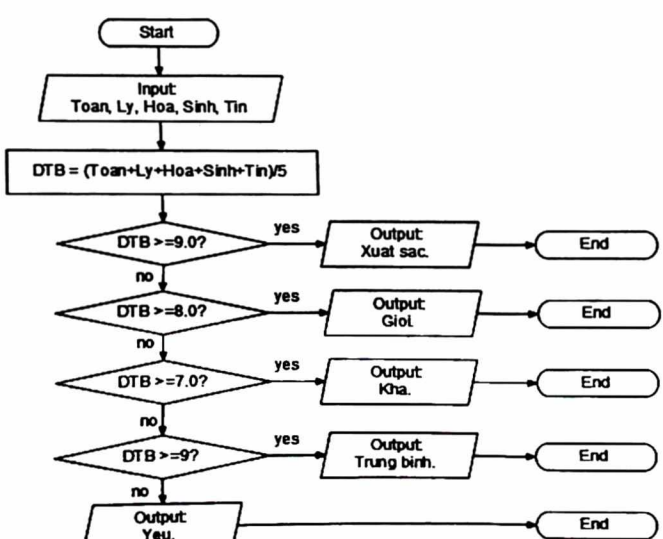
Bài tập 6. Xếp loại học sinh

Ý tưởng thuật toán

Ta có DTB = (Toan+Ly+Hoa+Sinh+Tin)/5.

Để xếp loại, ta sử dụng cấu trúc ***if .. elif..*** theo sơ đồ thuật toán sau:

Sơ đồ thuật toán



Chương trình

```
print("nhap diem Toan, Ly, Hoa, Sinh, Tin:")
Toan = float(input())
Ly   = float(input())
Hoa  = float(input())
Sinh = float(input())
Tin  = float(input())
DTB  = (Toan+Ly+Hoa+Sinh+Tin)/5
if DTB >= 9.0 :  print("Loai xuat sac.")
elif DTB >= 8.0:  print("Loai gioi.")
elif DTB >= 7.0:  print("Loai kha.")
elif DTB >= 5.0:  print("Loai trung binh.")
else:             print("Loai yeu.")
```

CHỦ ĐỀ 7. CẤU TRÚC LẶP

A. KIẾN THỨC CƠ BẢN

1. Lặp

Trong quá trình thực hiện chương trình, ta sẽ gặp nhiều công việc được thực hiện lặp đi lặp lại. Các công việc đó có thể lặp lại giống nhau y hệt hoặc chỉ khác nhau về giá trị tính toán, còn cách thức làm các công việc đó là giống nhau. Có hai dạng lặp đó là: 1) Lặp với số lần lặp được xác định trước và 2) Lặp với số lần lặp chưa xác định trước.

2. Câu lệnh for

Câu lệnh **for** cho phép thực hiện lặp với số lần đã xác định trước.

Cú pháp:

```
for <Tên biến> in [Tập các giá trị]:
    <Khối lệnh>
```

Thực hiện:

Giá trị biến sẽ lần lượt nhận các giá trị trong [Tập các giá trị], với mỗi lần nhận giá trị của biến sẽ là một lần thực hiện khối lệnh. Như vậy, số lần thực hiện khối lệnh (số lần lặp) bằng số các giá trị trong [Tập các giá trị].

Ví dụ 1. Chương trình:

```
for x in 1, 2, 5:
    print(x)
```

☞ Kết quả:

```
1
2
5
```

Chương trình trên còn được viết: `for x in 1, 2, 5: print(x)`

Ví dụ 2. Chương trình:

```
a = 10
b = 20
c = 30
s = 0
for x in a, b, c:
    s = s + x
print("Tong a + b + c bang: ", s)
```

☞ Kết quả:

```
Tong a + b + c bang:  60
```

Ví dụ 3. Chương trình:

```
for x in range(3):
    print("Python rat tuyen voi.")
```

☞ Kết quả:

```
Python rat tuyen voi.
Python rat tuyen voi.
Python rat tuyen voi.
```

Ví dụ 4. Chương trình:

```
for x in range(3):
    print("x = ",x)
```

☞ Kết quả:

```
x =  0
x =  1
x =  2
```

Ví dụ 5. Chương trình:

```
for x in range(2, 5):
    print("x = ",x)
```

☞ Kết quả:

```
x =  2
x =  3
x =  4
```

Ví dụ 6. Chương trình:

```
for x in range(2, 5, 2):
    print("x = ",x)
```

☞ Kết quả:

```
x =  2
x =  4
```

❖ ***Chú ý***

🖐 *Câu lệnh*:

```
for x in range(n):
    print(x)
```

Giá trị của biến x sẽ lần lượt nhận giá trị 0, 1, 2, .., n − 1. Như vậy, câu lệnh **print(x)** *sẽ thực hiện lặp n lần.*

🖐 *Câu lệnh*:

```
for x in range(a,b):
    print(x)
```

Giá trị của biến x sẽ lần lượt nhận giá trị từ $a, a+1, .., b-1$. *Như vậy, câu lệnh* **print(x)** *sẽ thực hiện lặp* $b - a$ *lần. Khi* $a > b$ *sẽ không thực hiện lần lặp nào.*

🖐 *Câu lệnh*:

```
for x in range(a,b,c):
    print(x)
```

Giá trị biến x sẽ lần lượt nhận: $a, a+c, a+2c, ...$ *Tức là a là giá trị khởi đầu, số sau bằng số kề trước cộng thêm c. Giá trị c luôn khác không. Khi* $c > 0$, *các số sẽ thuộc [a, b) và khi* $c < 0$ *thì các số sẽ thuộc (b, a].*

Như vậy: $range(a, b, 1) = range(a, b)$.

Ví dụ 7. Chương trình:

```
for x in range(4, 10, 0):
    print("x = ",x)
```

Chương trình sẽ báo lỗi:

```
ValueError: range() arg 3 must not be zero
```

Điều này có nghĩa là, tham số c (tham số thứ 3) luôn phải khác không.

Ví dụ 8. Chương trình:

```
for x in range(1, -3, -1):
    print("x = ",x)
```

☞ Kết quả:

```
x =  1
x =  0
x =  -1
x =  -2
```

3. Câu lệnh while

Câu lệnh while cho phép thực hiện lặp với số lần chưa xác định trước. Việc lặp được thực hiện khi có điều kiện thỏa mãn.

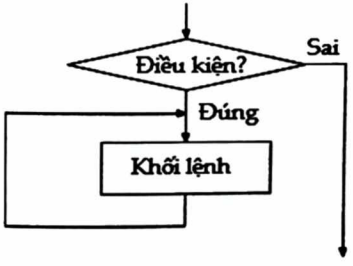
Cú pháp

While <Điều kiện>:

<Khối lệnh>

Thực hiện: Nếu <Điều kiện> đúng thì sẽ thực hiện <Khối lệnh>.

Sơ đồ thực hiện



Ví dụ 9. Chương trình:

```
n = 1
while n <= 3:
    print("n = ", n)
    n = n + 1
```

☛ Kết quả:

```
n =  1
n =  2
n =  3
```

Ví dụ 10. Chương trình:

```
n = 1
while n <= 3:
    print("n = ", n)
    n = n + 1
print("Ngoai vong lap while ")
```

☛ Kết quả:

```
n =  1
n =  2
n =  3
Ngoai vong lap while
```

❖ ***Nhận xét***

Khi sử dụng câu lệnh `while` *để thực hiện lặp, ta cần chú ý đến:*

- *Câu lệnh khởi tạo điều kiện:*

  Ở ví dụ trên là câu lệnh: $n = 1$;
- *Biểu thức điều kiện:*

  Ở ví dụ trên là câu lệnh: $n <= 3$;
- *Câu lệnh cập nhật điều kiện:*

  Ở ví dụ trên là câu lệnh: $n = n + 1$.

4. Câu lệnh continue

Câu lệnh **continue** được viết trong khối lệnh của các vòng lặp. Khi thực hiện câu lệnh **continue**, các câu lệnh phía sau trong khối lệnh sẽ không được thực hiện mà sẽ quay lại lần lặp tiếp theo.

Ví dụ 11. Chương trình:

```
for x in 1, 2, 3, 4, 5:
    if( x % 2 == 0):
        continue
    print("x = ",x," la so le")
```

☞ Kết quả:

```
x =  1  la so le
x =  3  la so le
x =  5  la so le
```

5. Câu lệnh break

Câu lệnh **break** được viết trong khối lệnh của vòng lặp. Khi thực hiện câu lệnh **break**, chương trình sẽ thoát khỏi vòng lặp trong cùng chứa nó.

Ví dụ 12. Chương trình:

```
for x in 1, 2, 3, 4, 5:
    if(x == 4):
        break
    print("x = ",x)
```

☞ Kết quả:

```
x =  1
x =  2
x =  3
```

Ví dụ 13. Chương trình:

```
for x in range(3):
    for y in range(6):
```

```
        if y == 3:
            break
        print("x , y:", x, y,sep = ' ')
```

☛ Kết quả:

```
x , y: 0 0
x , y: 0 1
x , y: 0 2
x , y: 1 0
x , y: 1 1
x , y: 1 2
x , y: 2 0
x , y: 2 1
x , y: 2 2
```

B. BÀI TẬP ÔN LUYỆN

Sử dụng cấu trúc lặp, hãy viết chương trình thực hiện các công việc sau:

Bài tập 1. Tổng các chữ số

Cho một số tự nhiên n ($n \leq 10^{1000}$). Tính tổng các chữ số của n. Ví dụ: n = 123, tổng các chữ số bằng $1 + 2 + 3 = 6$.

Bài tập 2. Chữ số lớn nhất và chữ số nhỏ nhất

Cho một số tự nhiên n ($n \leq 10^{1000}$). Tìm chữ số lớn nhất và chữ số nhỏ nhất của n. Ví dụ n = 123, chữ số lớn nhất bằng 3, chữ số nhỏ nhất bằng 1.

Bài tập 3. Ước chung lớn nhất và bội chung nhỏ nhất

Cho hai số nguyên dương a và b ($a, b \leq 10^9$). Tìm ước chung lớn nhất và bội chung nhỏ nhất của a và b. Ví dụ: $a = 9, b = 6$, *ước chung lớn nhất bằng 3, bội chung nhỏ nhất bằng 18.*

Bài tập 4. Kiểm tra số nguyên tố

Cho một số tự nhiên n ($n \leq 10^{12}$). Kiểm tra xem n có phải là một số nguyên tố hay không?

Bài tập 5. Số hoàn hảo

Một số tự nhiên n được gọi là một số hoàn hảo nếu tổng các ước dương của n bằng 2n. Ví dụ n = 6, có các ước dương: 1, 2, 3, 6; tổng các ước này bằng 12 và bằng 2n. Vậy 6 là một số hoàn hảo. Cho hai số tự nhiên a và b ($1 \leq a \leq b \leq 10000$). Tìm tất cả các số hoàn hảo thuộc [a; b].

Bài tập 6. Số lũy thừa 2 – 3

Một số tự nhiên n được gọi là số lũy thừa 2 − 3 nếu n có dạng: $n = 2^x 3^y$. Ví dụ, $n = 2, 3, 6, 12, ..$ là các số lũy thừa 2 − 3. Cho n, kiểm tra xem n có phải là số lũy thừa 2 − 3 hay không?

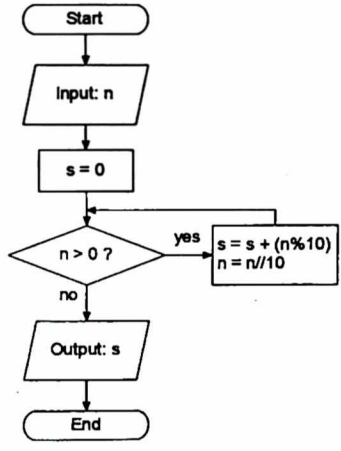
C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tổng chữ số

Ý tưởng thuật toán

Để tính tổng các chữ số, ta sẽ lặp để tách các chữ số của n ra rồi cộng chúng lại. Chữ số hàng đơn vị của n bằng $n\%10$ (số dư khi chia cho 10). Các chữ số còn lại tạo thành số $x = n//10$ (thương khi chia cho 10). Tiếp tục lấy chữ số hàng đơn vị của x chính là chữ số hàng chục của n. Việc lặp lại như vậy cho đến khi x bằng 0.

Sơ đồ thuật toán



Chương trình

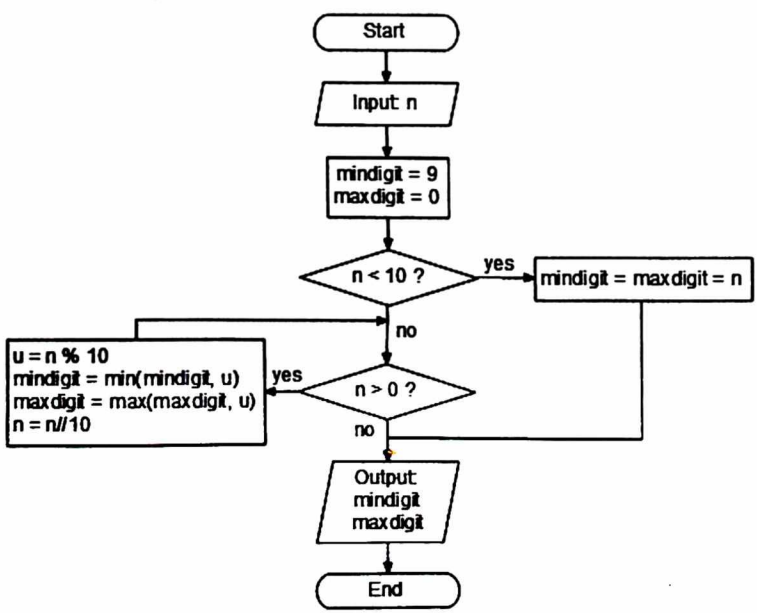
```
n = int(input("Nhap so tu nhien n:"))
s = 0
while n > 0:
    s = s + n%10
    n = n//10
print("Tong cac chu so cua n bang: ",s)
```

Bài tập 2. Chữ số lớn nhất, chữ số nhỏ nhất

Ý tưởng thuật toán

Việc tìm chữ số lớn nhất và chữ số nhỏ nhất của số tự nhiên n có thể làm được hoàn toàn bằng cách tách từng chữ số của n. Trong quá trình tách các chữ số của n, ta tìm được chữ số lớn nhất và chữ số nhỏ nhất.

Sơ đồ thuật toán



Chương trình

```
n = int(input("Nhap so tu nhien n: "))
mindigit = 9  # chu so lon nhat co the nhan
maxdigit = 0  # chu so nho nhat co the nhan
if n < 10:    # n co mot chu so
   maxdigit = mindigit = n
else:
   while n > 0:
      u = n%10
      maxdigit = max(maxdigit, u)
      mindigit = min(mindigit, u)
      n = n//10
print("Chu so nho nhat: ",mindigit)
print("Chu so lon nhat: ",maxdigit)
```

❖ ***Nhận xét***

Trong bài tập 1 và 2, số lần lặp để tách các chữ số ra bằng số các chữ số của n. Chính vì vậy chương trình sẽ chạy được với các số n rất lớn (có thể đến hàng triệu chữ số). Đây là một điểm mạnh của ngôn ngữ lập trình Python mà các ngôn ngữ như C++, Pascal,.. không có được.

Bài tập 3. Ước chung lớn nhất và bội chung nhỏ nhất

• *Ước chung lớn nhất*

Thuật toán Euclide:

Ta có: Nếu $b = 0$ thì $(a, b) = a$, ngược lại, thực hiện phép chia Euclide:

$a = bq + r$; (a chia cho b được thương là q và dư r), thì có $(a, b) = (b, r)$.

Nếu r = 0 thì $(a, b) = (b, r) = (b, 0) = b$, nếu $r \neq 0$, ta thay $a \leftarrow b, b \leftarrow r$ và tiếp tục thực hiện phép chia Euclide như trên.

Ví dụ: $a = 9, b = 6$.

Phép chia Eucide: $a = bq + r$	**ƯCLN(a, b)**
$9 = 6 \times 1 + 3$	$(9,6) = (6,3)$
$6 = 3 \times 2 + 0$	$(6,3) = (3,0) = 3$

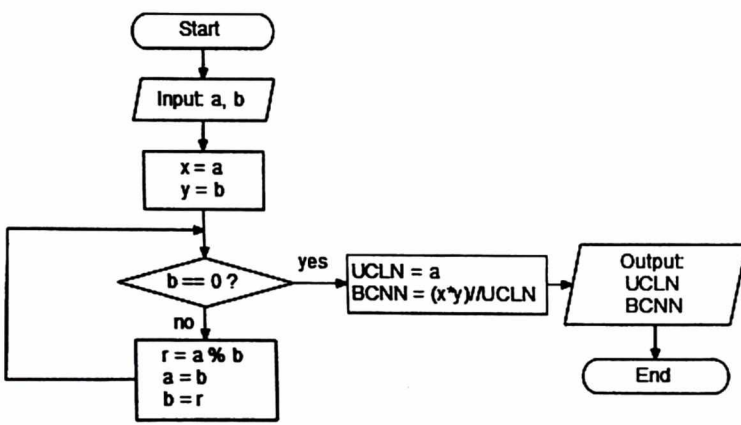
Như vậy (9, 6) = 3.

• *Bội chung nhỏ nhất*

Ta có công thức tính bội chung nhỏ nhất của hai số nguyên dương là.

$BCNN(a,b) = \frac{a \times b}{(a,b)}$.

Sơ đồ thuật toán



Chương trình

```
a = int(input("nhap so nguyen duong a: "))
b = int(input("nhap so nguyen duong b: "))
#tim uoc chung lon nhat theo thuat toan Euclide
x = a
y = b
while b > 0:
```

```
        r = a%b
        a = b
        b = r
    UCLN = a
    BCNN = x*y//UCLN
    print("UCLN(a, b) bang: ", UCLN)
    print("BCNN(a, b) bang: ", BCNN)
```

Bài tập 4. Kiểm tra số nguyên tố

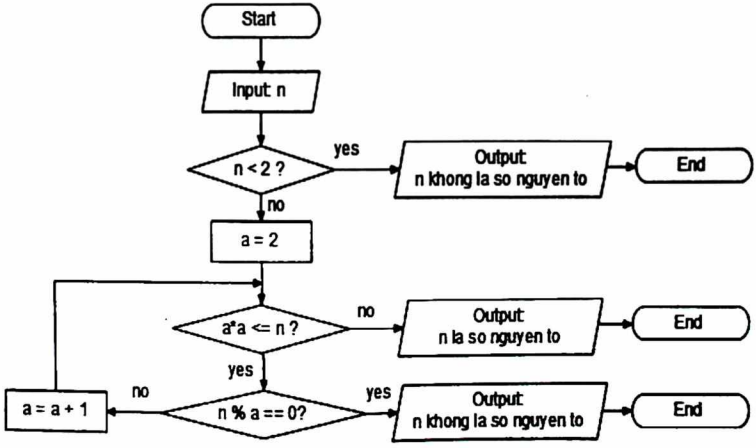
Ý tưởng thuật toán:

Ta nhận thấy rằng, một số tự nhiên n không phải là số nguyên tố nếu n có một ước khác 1 và n, nghĩa là n có thể phân tích: $n = a \times b$ với $2 \leq a < n$. Điều này cũng suy ra được $2 \leq b < n$. Ta giả sử $a \leq b$, suy ra:

$a \times a \leq a \times b = n \rightarrow 2 \leq a \leq \sqrt{n}$.

Như vậy, ta sẽ lần lượt kiểm tra $a = 2, 3, 4, .., [\sqrt{n}]$, nếu có giá trị a thỏa mãn $n : a$ thì kết luận n không phải là số nguyên tố, ngược lại, n ($n > 1$) sẽ là một số nguyên tố.

Sơ đồ thuật toán



Chương trình

```
n = int(input("Nhap so tu nhien n: "))
if(n < 2): print("n khong phai la so nguyen to.")
else:
   a = 2
   NT = True
   while a*a <= n:
      if n%a == 0:
         NT = False
         break
      a = a + 1
   if NT: print("n la so nguyen to.")
   else: print("n khong phai la so nguyen to.")
```

Bài tập 5. Số hoàn hảo

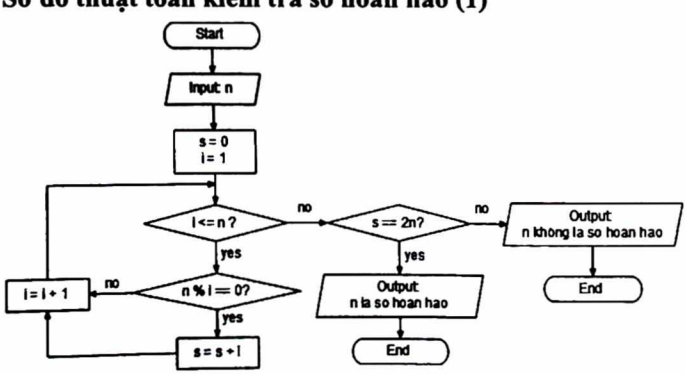
Ý tưởng thuật toán

Trong bài này, ta sẽ xét (duyệt) tất cả các số $n \in [a; b]$, với mỗi số n, kiểm tra xem n có phải là một số hoàn hảo hay không?

Để kiểm tra n có phải là số hoàn hảo, ta tính tổng các ước dương của n.

Cách 1. Xét tất cả các số $i = 1, 2, .., n$, với mỗi i, nếu là ước của n sẽ cộng giá trị i vào tổng ước.

Sơ đồ thuật toán kiểm tra số hoàn hảo (1)



Chương trình

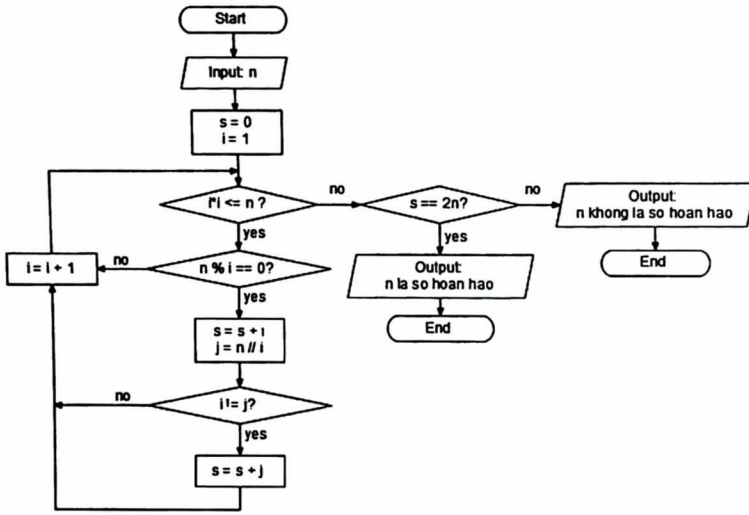
```
n = int(input("Nhap so tu nhien n: "))
s = 0
for i in range(1, n+1):
    if n%i == 0:
        s = s + i
if s == 2*n: print(n," la so hoan hao")
else: print(n," khong la so hoan hao")
```

Cách 2. Nhận thấy nếu i là ước của n thì sẽ có $n = i \times j$. Như vậy, khi biết i, n thì tính được j và j cũng là một ước của n. Không mất tính tổng quát, giả sử $i \leq j$, suy ra $i \leq \sqrt{n}$. Từ đó, ta xét các giá trị $i = 1, 2, .., [\sqrt{n}]$, với mỗi giá trị i là ước của n thì cộng i vào tổng ước và tính $j = n//i$, nếu $i \neq j$ thì j là một ước khác và cộng j vào tổng ước.

Chương trình

```
n = int(input("Nhap so tu nhien n: "))
s = 0
can_bac_2 = int(n**(1/2))
for i in range(1, 1 + can_bac_2):
    if n%i == 0:
        s = s + i
        j = n//i
        if (j != i): s = s + j
if s == 2*n: print(n," la so hoan hao")
else: print(n," khong la so hoan hao")
```

Sơ đồ thuật toán kiểm tra số hoàn hảo (2)



❖ ***Nhận xét***

Trong cách 1, vòng lặp: `for i in range(1, n+1):` sẽ thực hiên lặp n lần.

Trong cách 2, vòng lặp: `for i in range(1, 1 + can_bac_2):` sẽ thực hiện lặp trong $\sqrt{n}$ lần.

Số lần lặp trong cách 1 và cách 2 chênh nhau rất lớn khi n lớn, chẳng hạn, khi n = 1.000.000, thì cách 1 sẽ lặp 1.000.000 lần và cách 2 sẽ lặp 1.000 lần.

Chương trình

```
a = int(input("Nhap so tu nhien a: "))
b = int(input("Nhap so tu nhien b: "))
for n in range(a, b + 1):
    s = 0
    can_bac_2 = int(n**(1/2))
    for i in range(1, 1 + can_bac_2):
        if n%i == 0:
```

```
            s = s+i
            j = n//i
            if j != i: s = s + j
    if(s == 2*n):
        print(n)
```

Bài tập 6. Số lũy thừa $2-3$

Ý tưởng thuật toán

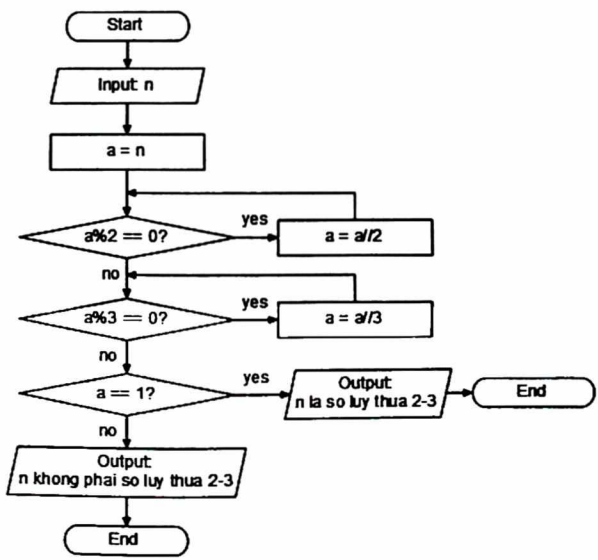
Ta nhận thấy, một số lũy thừa $2-3$ có dạng $n = 2^x 3^y$, vì vậy, khi thực hiện x lần chia n cho 2 và thực hiện y lần chia n cho 3 thì ta được kết quả bằng 1. Do đó, với số nguyên dương n bất kì, ta thực hiện:

Nếu n chia hết cho 2 thì $n = n//2$, thực hiện cho đến khi n không chia hết cho 2.

Nếu n chia hết cho 3 thì $n = n//3$, thực hiện cho đến khi n không chia hết cho 3.

Thực hiện xong, nếu n bằng 1 thì giá trị ban đầu là số lũy thừa $2-3$.

Sơ đồ thuật toán



Chương trình

```
n = int(input("Nhap so nguyen duong n:"))
a = n
while a%2 == 0: a = a//2
while a%3 == 0: a = a//3
if a==1:
   print(n, " la so luy thua 2-3")
else:
   print(n, " khong la so luy thua 2-3")
```

CHỦ ĐỀ 8. KIỂU DỮ LIỆU CHUỖI

. KIẾN THỨC CƠ BẢN

Định nghĩa chuỗi

Chuỗi là một dãy các kí tự. Mọi kí tự nằm trong " " (ngoặc kép) ı ' ' (ngoặc đơn) đều được xem là chuỗi trong Python. Chuỗi rỗng là ıuỗi không có kí tự nào.

Ví dụ về chuỗi:

"Day la mot chuoi!", 'Day cung la mot chuoi!'.

Các kí tự trong chuỗi là các kí tự thuộc bảng mã Unicode hoặc ıng mã ASCII. Chính vì vậy, chuỗi trong Python có hỗ trợ tiếng Việt. í tự trong chuỗi được đánh chỉ số thứ tự bắt đầu bằng chỉ số 0, 1, .. ıeo hướng từ trái sang phải) và bắt đầu từ -1, -2, .. (theo hướng từ ıải sang trái).

Số kí tự trong chuỗi được gọi là độ dài của chuỗi đó.

Xét chuỗi "Tu hoc Python". Cách đánh chỉ số của các kí tự như sau:

	T	u		h	o	c		P	y	t	h	o	n
:hỉ	0	1	2	3	4	5	6	7	8	9	10	11	12
ố	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Ví dụ 1. Chương trình đơn giản về chuỗi:

```
str1 = "Tu hoc Lap trinh Python"
str2 = 'Tu hoc Lap trinh Python'
print(str1)
print(str2)
```

☞ Kết quả:

```
Tu hoc Lap trinh Python
Tu hoc Lap trinh Python
```

Ví dụ 2. Chương trình:

```
str1 = "Tu hoc 'Lap trinh Python'"
str2 = 'Tu hoc "Lap trinh Python"'
print(str1)
print(str2)
```

☞ Kết quả:

```
Tu hoc 'Lap trinh Python'
Tu hoc "Lap trinh Python"
```

2. Thao tác với chuỗi

a) Hàm lấy độ dài

len(*biến chuỗi*)

Ví dụ 3. Chương trình:

```
str1 = "abcde"
print("Chuoi: ",str1)
print("Do dai: ",len(str1))
```

☞ Kết quả:

```
Chuoi:  abcde
Do dai:  5
```

b) Truy cập đến từng kí tự trong chuỗi

Để truy cập đến một kí tự trong chuỗi, ta thực hiện:

***<biến chuỗi>*[*chỉ số*].**

Như vậy nếu đặt n là độ dài của chuỗi, thì các chỉ số của kí tự được đánh từ $0, 1, .., n-1$ (theo hướng từ trái sang phải) và đánh chỉ số từ -1, -2, .., $-n$ (theo hướng từ phải sang trái).

Ví dụ 4. Chương trình:

```
str = "abc"
n = len(str)
```

```
print(str[0]," == ",str[-n])
print(str[1]," == ",str[-n+1])
print(str[2]," == ",str[-n+2])
print("Tat ca cac ki tu trong chuoi:")
for i in range(n):
    print(str[i])
```

☞ Kết quả:

```
a  ==  a
b  ==  b
c  ==  c
Tat ca cac ki tu trong chuoi:
a
b
c
```

c) Ghép chuỗi (cộng chuỗi) và lặp chuỗi

❖ Ghép chuỗi

Python cho phép tạo chuỗi mới bằng cách ghép các chuỗi lại với nhau:

<Chuỗi 1> + ***<Chuỗi 2>*** + .. + ***<Chuỗi n>***

Trả về chuỗi được tạo thành bằng cách ghép *<Chuỗi 1>*, *<Chuỗi 2>*, .. *<Chuỗi n>* theo thứ tự lại với nhau.

Ví dụ về ghép chuỗi:

"Tu hoc " + "Lap trinh Python" = "Tu hoc Lap trinh Python"

Ví dụ 5. Chương trình:

```
str1 = "Tu hoc "
str2 = "Lap trinh Python"
str = str1 + str2
print(str)
```

☛ Kết quả:

```
Tu hoc Lap trinh Python
```

❖ **Lặp chuỗi**

Để ghép nhiều lần một chuỗi lại với nhau, ngoài cách ghép trực tiếp, Python còn cho phép thực hiện ngắn gọn hơn:

<biến chuỗi><số lần ghép>*

<số lần ghép><biến chuỗi>*

Ví dụ 6. Chương trình lặp chuỗi:

```
st = "Python"
str1 = st*8
str2 = 8*st
print(str1)
print(str2)
```

☛ Kết quả:

```
PythonPythonPythonPythonPythonPythonPythonPython
PythonPythonPythonPythonPythonPythonPythonPython
```

d) So sánh chuỗi

So sánh chuỗi được thực hiện bằng cách lần lượt so sánh các kí tự từ trái sang phải. Nếu gặp một cặp kí tự không bằng nhau thì chuỗi nào chứa kí tự lớn hơn sẽ là chuỗi lớn hơn. Nếu một chuỗi là phần đầu của chuỗi kia thì chuỗi đó là nhỏ hơn.

Ví dụ: "abc" < "abcd"; "acbb" > "abeeeeeeee"

Ví dụ 7. Chương trình:

```
str1 = "abcXy"
str2 = "aef"
if str1 < str2:
   print("Xau nho hon: ",str1)
else:
   print("Xau lon hon: ",str2)
```

☞ Kết quả:

```
Xau nho hon:  abcXy
```

e) Lấy chuỗi con

Để lấy chuỗi con gồm các kí tự liên tiếp từ chỉ số a đến chỉ số $b-1$ ($a \leq b$) ta thực hiện:

<biến chuỗi>[a: b]

Ví dụ 8. Chương trình:

```
str = "abcdefg"
st0_3 = str[0:3]
st1_5 = str[1:5]
print("Chuoi con tu chi so 0 den 2: ", st0_3)
print("Chuoi con tu chi so 1 den 4: ", st1_5)
```

☞ Kết quả:

```
Chuoi con tu chi so 0 den 2:  abc
Chuoi con tu chi so 1 den 4:  bcde
```

❖ ***Chú ý***

- *Lấy a kí tự đầu của chuỗi, <biến chuỗi>[: a], tức là cũng bằng câu lệnh <biến chuỗi>[0: a].*
- *Lấy các kí tự từ chỉ số a đến cuối chuỗi, <biến chuỗi>[a:].*

  Như vậy <biến chuỗi>[: a] + <biến chuỗi>[a:] == <biến chuỗi>
- *<biến chuỗi>*[:] == *<biến chuỗi>*

Ví dụ 9. Chương trình:

```
str = "abcde"
print("3 ki tu dau: ", str[:3])
print("Cac ki tu tu chi so 3 den cuoi: ",
str[3:])
```

☞ Kết quả:

```
3 ki tu dau:  abc
Cac ki tu tu chi so 3 den cuoi:  de
```

❖ **Dạng tham số**

<biến chuỗi>[*a*: *b*: *c*]

Trả về chuỗi gồm các kí tự trong *<biến chuỗi>* có chỉ $a, a+c, .., a+tc$ với $a+tc < b$.

Ví dụ 10. Chương trình:

```
st = "abcdefgh"
str1 = st[0:6:2]
str2 = st[6:0:-1]
print(str1)
print(str2)
```

☛ Kết quả:

```
ace
gfedcb
```

f) Đảo ngược thứ tự chuỗi

Để đảo ngược thứ tự của một chuỗi, ta thực hiện:

<biến chuỗi>[::−1]

Ví dụ 11. Chương trình:

```
st = "abcdefg"
str1 = st[::-1]
print(str1)
```

☛ Kết quả:

```
gfedcba
```

3. Một số nhận xét

- Trong Python không có kiểu kí tự như nhiều ngôn ngữ trình khác. Mỗi kí tự trong Python được xem như một chuỗi độ dài bằng 1.

- Khi khai báo một biến chuỗi, ta không thể thay đổi các kí tự trong biến chuỗi đó.

Ví dụ 12. Chương trình lỗi về thay đổi kí tự:

```
1  str1 = "abc"
2  str1[0] = "A"
```

```
3  print(str1)
```

- Để kết nối chuỗi với số hoặc các kiểu dữ liệu khác, ta có thể thực hiện ép kiểu như sau:

  str(*biến kiểu số*)

Ví dụ 13. Chương trình lỗi về ghép kiểu chuỗi với kiểu số:

```
1  str1 = "So hoc sinh lop 10A la: "
2  n = 35
3  str2 = str1 + n
```

```
4  print(str2)
```

Ví dụ 14. Chương trình về ép kiểu:

```
str1 = "So hoc sinh lop 10A la: "
n = 35
str2 = str1 + str(n)
print(str2)
```

☞ Kết quả:

```
So hoc sinh lop 10A la: 35
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tách kí tự số và kí tự chữ cái

Nhập vào một chuỗi gồm các kí tự số và kí tự chữ cái La-tinh thường 'a', 'b', .. 'z'. Hãy tách chuỗi được nhập vào thành hai chuỗi,

một chuỗi chỉ gồm các kí tự chữ số, chuỗi còn lại gồm các kí tự chữ cái. Ví dụ: "123dffg3d" được tách thành hai chuỗi: "1233" và "dffgd".

Bài tập 2. Chuỗi đối xứng

Một chuỗi được gọi là chuỗi đối xứng nếu đọc chúng từ trái sang phải cũng giống như đọc từ phải sang trái. Ví dụ: "madam","1221" là các chuỗi đối xứng, chuỗi "12334" không phải là một chuỗi đối xứng.

Viết chương trình nhập vào một chuỗi và kiểm tra xem có phải là chuỗi đối xứng hay không?

Bài tập 3. Nén chuỗi

Cho một chuỗi str1 gồm các kí tự chữ cái La-tinh thường. Ta thực hiện nén chuỗi theo quy tắc sau: với một dãy các kí tự giống nhau liên tiếp, ta sẽ thay thế dãy này bằng: <kí tự><số lần xuất hiện>.

Ví dụ: str1 = "aaabccccddda"; dãy "aaa" thay thế bằng "a3"; dãy "ccccc" thay thế bằng "c4"; dãy "ddd" thay thế bằng "d3". Do vậy "aaabccccddda" được nén thành "a3bc4d3a".

Chú ý: Ta không nén "aaa" thành "a2a","b" thành "b1".

Yêu cầu: Cho chuỗi str1, hãy đưa ra chuỗi nén được.

Bài tập 4. Giải nén chuỗi

Cho chuỗi str2 được nén bởi chuỗi str1 theo quy tắc trên (bài tập 3). Hãy đưa ra chuỗi str1.

Bài tập 5. Mật khẩu

Một phần mềm yêu cầu đặt mật khẩu an toàn là chuỗi kí tự có đủ 3 loại kí tự:

- *Kí tự chữ số: '0', '1', .., '9'.*
- *Kí tự La-tinh thường: 'a', 'b', .. 'z'.*
- *Kí tự La-tinh hoa: 'A', 'B', .. 'Z'*

Cho chuỗi kí tự str1 chỉ gồm các kí tự thuộc 3 loại kí tự trên. Tìm một chuỗi con gồm các kí tự liên tiếp của str1 sao cho: Có thể làm được mật khẩu và có độ dài nhỏ nhất. Ví dụ: str1 = "aBBcB1BCc1", chuỗi con có thể làm được mật khẩu và có độ dài nhỏ nhất là "cB1".

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tách kí tự số và kí tự chữ cái

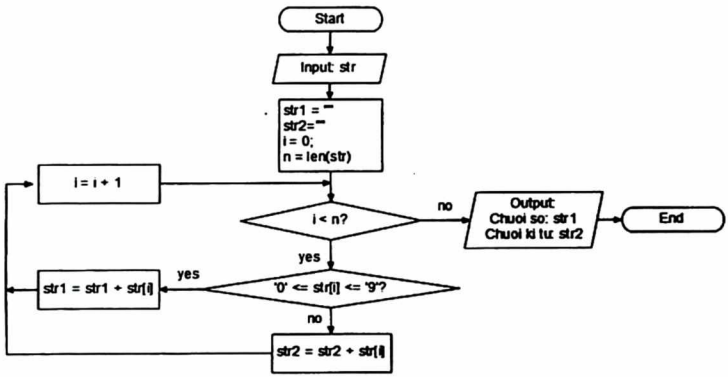
Ý tưởng thuật toán

Ta sử dụng chuỗi *str*1 để chứa chuỗi gồm các kí tự số, chuỗi *str*2 để chứa các kí tự chữ cái La-tinh thường (ban đầu chuỗi *str*1, *str*2 được là chuỗi rỗng).

Lần lượt xét các kí tự của chuỗi nhập vào, khi gặp kí tự là kí tự số thì ghép chúng vào chuỗi *str*1, ngược lại (kí tự La-tinh thường) thì ghép chúng vào xâu *str*2.

Kí tự *u* là một kí tự số nếu: '0' <= *u* <= '9'.

Sơ đồ thuật toán



Chương trình

```
str = input("Nhap chuoi gom ki tu so va chu cai
latinh: ")
str1 = str2 = ""
for u in str:
   if '0'<= u <= '9':
      str1 = str1 + u
   else:
      str2 = str2 + u
```

```
print("Xau gom cac ki tu so:  ",str1)
print("Xau gom cac ki tu chu: ",str2)
```

Bài tập 2. Chuỗi đối xứng

Ý tưởng thuật toán

Xét chuỗi $str = str_1 str_2 .. str_{n-1}$ với n là độ dài của chuỗi.

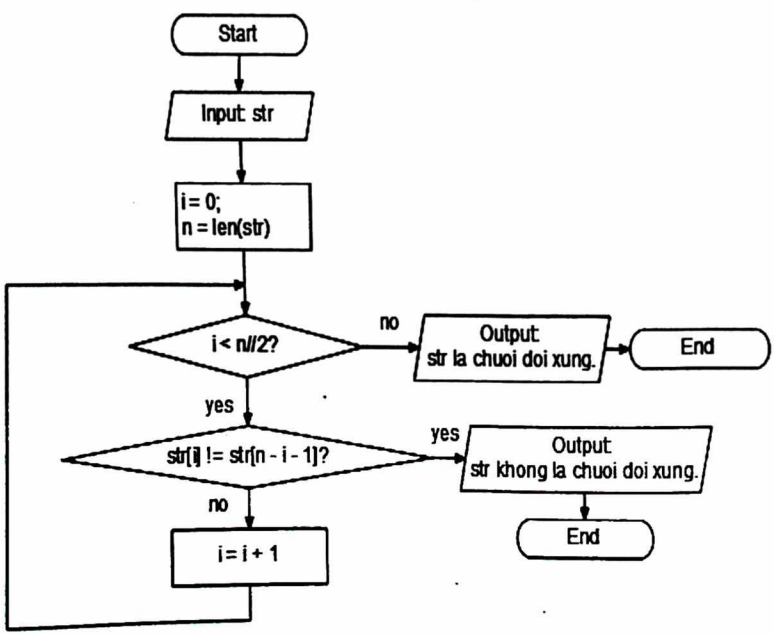
Nhận thấy, chuỗi str là chuỗi đối xứng nếu $str_i = str_{n-i-1}$ với $i = 0, 1, 2, .., \frac{n}{2}$.

Do vậy, ta sẽ kiểm tra các cặp kí tự str_i và str_{n-i-1} với $i = 0, 1, .., \frac{n}{2}$.

Nếu có một chỉ số i mà str_i khác str_{n-i-1} thì chuỗi str không phải là chuỗi đối xứng.

Nếu str_i bằng str_{n-i-1} với mọi $i = 0, 1, .., \frac{n}{2}$ thì chuỗi str là chuỗi đối xứng.

Sơ đồ thuật toán



Chương trình

```
str = input("Nhap chuoi, kiem tra doi xung:")
n = len(str)              #lay do dai chuoi
dx = True .               #danh dau trang thai
for i in range(n//2):
    if str[i] != str[n-i-1]:
      dx = False          #xac nhan lai trang thai
      break
if dx == True:
    print("Chuoi doi xung")
else:
    print("Khong phai la chuoi doi xung")
```

Một lời giải đẹp!

Với *n* chẵn, ta chia chuỗi *str* thành hai chuỗi có độ dài bằng nhau tương ứng là *str*1 và *str*2.

Với *n* lẻ, *str*1 là chuỗi bên trái phần tử chính giữa, *str*2 là chuỗi bên phải phần tử chính giữa.

Khi đó *str* là chuỗi đối xứng nếu *str*1 bằng chuỗi đảo ngược của *str*2.

Chương trình

```
str = input("Nhap chuoi, kiem tra doi xung: ")
n = len(str)
str1 = str[:(n//2)]
str2 = str[(n+1)//2:]
str2 = str2[::-1]
if str1 == str2:
    print("chuoi doi xung")
else:
    print("chuoi khong doi xung")
```

Bài tập 3. Nén chuỗi

Ý tưởng thuật toán

Đặt n = ***len***($str1$). Chuỗi kết quả st bằng rỗng.

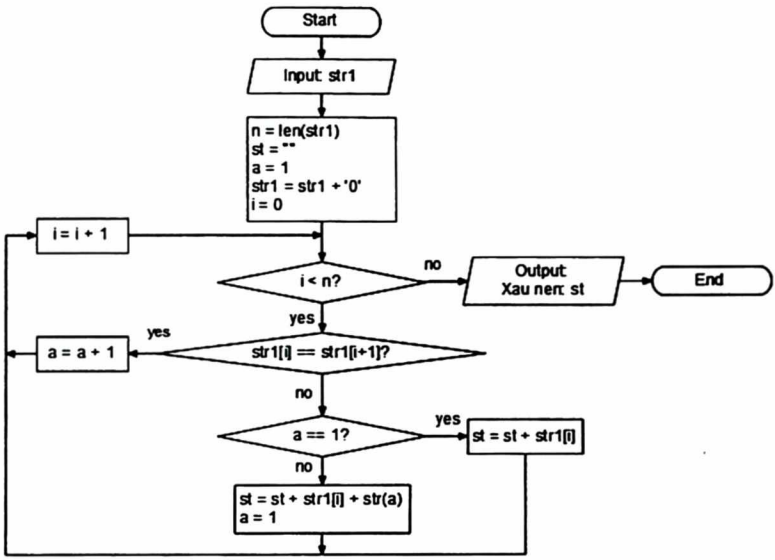
Duyệt các kí tự của chuỗi, từ kí tự $str1[0]$ đến $str1[n-1]$. Quá trình duyệt, ta sẽ tìm được dãy các kí tự liên tiếp bằng nhau. Ứng với mỗi dãy các kí tự liên tiếp bằng nhau, ta nén chúng theo yêu cầu đề bài và thêm vào chuỗi kết quả st.

Vị trí i là vị trí kết thúc một dãy các kí tự liên tiếp bằng nhau nếu $str1[i] != str1[i+1]$ và khi đó vị trí bắt đầu một dãy kí tự liên tiếp mới là $i+1$. Chú ý là, trừ trường hợp ngoại lệ đó là $i = n-1$ (là vị trí cuối cùng trong chuỗi). Để khắc phục điều này, ta sử dụng **kĩ thuật cầm canh**, thêm vào cuối chuỗi một kí tự khác $str1[n-1]$, chẳng hạn kí tự '0'. Dãy các kí tự liên tiếp bằng nhau kết thúc tại vị trí i được nén thành $str1[i]$ + ***str***(a) với a ($a > 1$) là số kí tự trong dãy liên tiếp đó.

Chương trình

```
str1 = input("Nhap xau de nen:")
st = ""                    #chuoi ket qua, ban dau rong
a = 1                      #so ki tu trong day ki tu
lien tiep
n = len(str1)
str1 = str1 + '0'          #cam canh ki tu '0'
for i in range(n):
   if str1[i] == str1[i+1]:
      a = a + 1
   elif a == 1:    #ket thuc day ki tu lien tiep
bang nhau
      st = st + str1[i];
   else:
      st = st + str1[i]+ str(a)
      a = 1
print("Chuoi nen: ",st)
```

Sơ đồ thuật toán



Bài tập 4. Giải nén chuỗi

Ý tưởng thuật toán

Đặt $n = len(str2)$, chuỗi kết quả $str1 = ""$.

Nhận thấy, mỗi đoạn liên tiếp các kí tự giống nhau được nén thành dạng:

<kí tự><số lần xuất hiện>.

Ví dụ: a12 → a là kí tự, số lần xuất hiện 12.

Đặt *<kí tự>* = ch, *<số lần xuất hiện>* = a. Ta đi tìm ch và a, khi đó dãy các kí tự liên tiếp giống nhau là: $ch * a$.

Để thực hiện điều này, ta duyệt lần lượt các kí tự trong chuỗi (từ chỉ số $n - 1$ về chỉ số 0) và sử dụng chuỗi: $so_$ chứa các kí tự số của *<số lần xuất hiện>*. Ban đầu $so_$ là chuỗi rỗng.

Với mỗi kí tự $str2[i]$:

Nếu $str2[i]$ là kí tự số thì: $so_ = str2[i] + so_$

Nếu $str2[i]$ là kí tự chữ cái thì cập nhật chuỗi $str1$.

- Nếu $so_$ là rỗng thì $str1 = str2[i] + str1$.
- Nếu so_ khác rỗng thì:

$a = (int)\, so_$

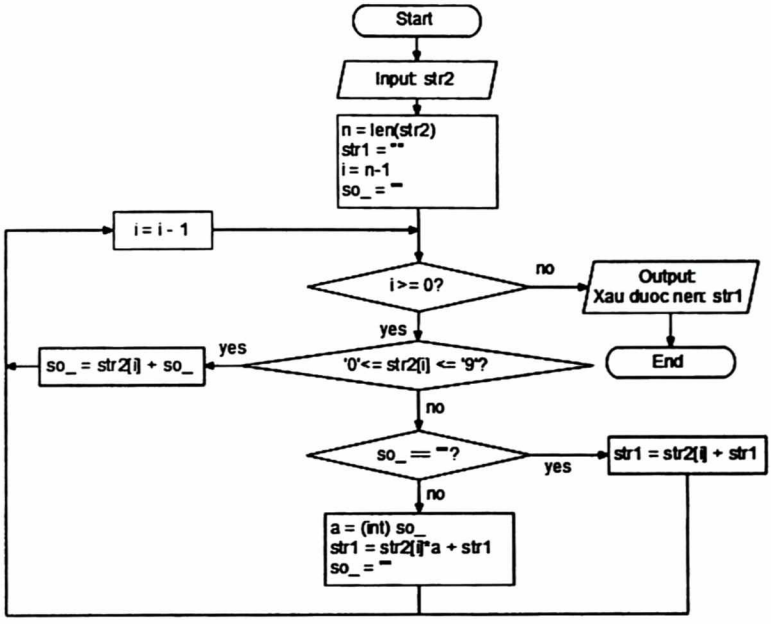
$str1 = str2[i] * a + str1$

$so_ = ""$

Chương trình

```
str2 = input("Nhap xau de giai nen:")
str1 = ""
so_ = ""
n = len(str2)
i = n - 1
while i >= 0:
    if "0" <= str2[i] <= "9":
        so_ = str2[i] + so_
    elif so_ == "":
        str1 = str2[i] + str1
    else:
        str1 = str2[i]*int(so_) + str1
        so_ = ""
    i = i - 1
print("Chuoi duoc nen: ",str1)
```

Sơ đồ thuật toán



Bài tập 5. Mật khẩu

Ý tưởng thuật toán

Đặt $n = len(str1)$. Lần lượt duyệt qua các kí tự từ chỉ số 0 đến chỉ số $n-1$. Tại mỗi chỉ số i, tìm các chỉ số *index_hoa*, *index_thuong*, *index_so* tương ứng là chỉ số của kí tự hoa, kí tự thường, kí tự số gần $str1[i]$ nhất. Chú ý là, các chỉ số *index_hoa*, *index_thuong*, *index_so* nhỏ hơn hoặc bằng i. Ban đầu các chỉ số này được khởi tạo bằng -1. Khi đó, dãy các kí tự liên tiếp ngắn nhất kết thúc tại $str1[i]$ có thể làm được mật khẩu là $str1[t]..str1[i]$, với $t = min(index_hoa, index_thuong, index_so)$ và $t \neq -1$.

Ta sẽ tìm được dãy $str1[t]..str1[i]$ ngắn nhất khi duyệt hết các giá trị i.

Việc tìm các chỉ số *index_hoa*, *index_thuong*, *index_so* sẽ được thực hiện khi tại mỗi vị trí chỉ số i:

Nếu $str1[i]$ là kí tự hoa thì $index_hoa = i$;

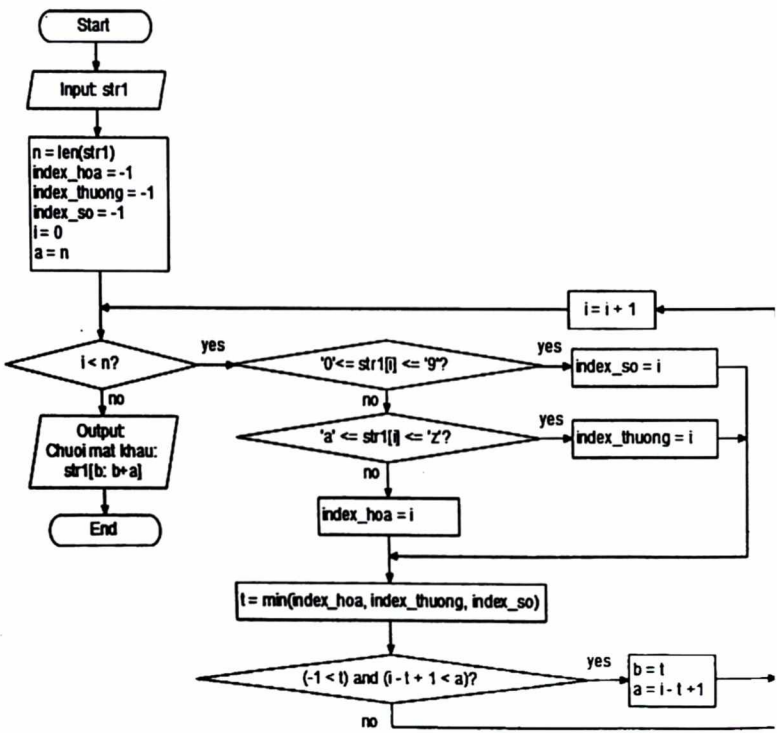
Nếu $str1[i]$ là kí tự thường thì $index_thuong = i$;

Nếu $str1[i]$ là kí tự số thì $index_so = i$;

Đưa ra chuỗi con ngắn nhất làm mật khẩu?

Để đưa ra chuỗi con làm mật khẩu, cần tìm độ dài ngắn nhá (giả sử là a) và chỉ số của kí tự đầu tiên (giả sử là b). Chuỗi cần tìm l $str1[b: b + a]$.

Sơ đồ thuật toán



Chương trình

```
str1 = input("Nhap chuoi str1 de tim chuoi mat
khau: ")
n = len(str1)
```

```
index_hoa = index_thuong = index_so = -1
a = n
for i in range(n):
   if '0' <= str1[i] <= '9':
      index_so = i
   elif 'a' <= str1[i] <= 'z':
      index_thuong = i
   else:
      index_hoa = i
   t = min(index_so, index_thuong, index_hoa)
   if -1 < t and i - t + 1 < a:
       b = t
       a = i - t + 1
print("Chuoi mat khau: ",str1[b:b+a])
```

CHỦ ĐỀ 9. MỘT SỐ THAO TÁC TRÊN CHUỖI

A. KIẾN THỨC CƠ BẢN

1. Tách chuỗi

Phương thức: <*biến chuỗi*>.split()

Tách chuỗi thành danh sách các chuỗi con mà kí tự phân các kí tự trắng (white space).

Ví dụ 1. chương trình:

```
str1 = "Tu hoc lap trinh Python"
L = str1.split()
print("Chuoi duoc tach thanh:")
print(L)
```

Kết quả:

```
Chuoi duoc tach thanh:
['Tu', 'hoc', 'lap', 'trinh', 'Python']
```

❖ Một số dạng có tham số

Kí tự phân tách mặc định là kí tự trắng, Python cho phép lập trình thay đổi kí tự phân tách này.

a) Tham số kí tự phân tách

<*biến chuỗi*>.split(<*kí tự phân tách*>)

Ví dụ 2. Chương trình:

```
str1 = "Tu hoc lap trinh, Python"
L = str1.split(",")
print("Chuoi duoc tach thanh:")
print(L)
```

Kết quả:

```
Chuoi duoc tach thanh:
['Tu hoc lap trinh', ' Python']
```

b) Tham số về số chuỗi tối đa được tách

<biến chuỗi>.**split**(*<kí tự phân tách>, <số chuỗi tối đa được tách>*)

Ví dụ 3. Chương trình:

```
str1 = "Tu#hoc#lap#trinh#Python"
L = str1.split("#", 2)
print("Chuoi duoc tach thanh:")
print(L)
```

☞ Kết quả:

```
Chuoi duoc tach thanh:
['Tu', 'hoc', 'lap#trinh#Python']
```

Ta nhận thấy, kết quả được tách ra làm 2 chuỗi con và chuỗi phần còn lại.

❖ ***Chú ý***

Tham số mặc định là -1, nghĩa là tách tất cả.

Ví dụ 4. Chương trình:

```
str1 = "Tu#hoc#lap#trinh#Python"
L = str1.split("#", -1)
print("Chuoi duoc tach thanh:")
print(L)
```

☞ Kết quả:

```
Chuoi duoc tach thanh:
['Tu', 'hoc', 'lap', 'trinh', 'Python']
```

2. Chuyển đổi kí tự hoa, kí tự thường

Phương thức: *<biến chuỗi>*.**upper()**

Trả về chuỗi sau khi chuyển đổi các kí tự trong chuỗi thành kí tự hoa.

- Phương thức: <*biến chuỗi*>.`lower()`

Trả về chuỗi sau khi chuyển đổi các kí tự trong chuỗi thành kí tự thường.

Ví dụ 5. Chương trình:

```
st = "abCDe123"
st_hoa = st.upper()
st_thuong = st.lower()
print("Chuoi hoa: ",st_hoa)
print("Chuoi thuong: ",st_thuong)
```

☞ Kết quả:

```
Chuoi hoa:  ABCDE123
Chuoi thuong:  abcde123
```

- Phương thức: <*biến chuỗi*>.`capitalize()`

Trả về chuỗi sau khi chuyển đổi kí tự đầu tiên trong chuỗi thành kí tự hoa.

Ví dụ 6. Chương trình:

```
str1 = "tu hoc lap trinh python"
st = str1.capitalize()
print(st)
```

☞ Kết quả:

```
Tu hoc lap trinh python
```

- Phương thức: <*biến chuỗi*>.`title()`

Trả về chuỗi sau khi chuyển đổi các kí tự đầu tiên của các từ thành kí tự hoa.

Ví dụ 7. Chương trình:

```
str1 = "Tu hoc lap trinh python"
st = str1.title()
print(st)
```

☞ Kết quả:

```
Tu Hoc Lap Trinh Python
```

➧ Phương thức: <*biến chuỗi*>.**swapcase()**

Trả về chuỗi sau khi chuyển các kí tự hoa thành kí tự thường, kí tự thường thành kí tự hoa.

Ví dụ 8. Chương trình:

```
str1 = "Tu Hoc Lap Trinh Python"
st = str1.swapcase()
print(st)
```

☞ Kết quả:

```
tU hOC lAP tRINH pYTHON
```

❖ ***Chú ý***

Sau thi thực hiện các phương thức: str1.*upper*(), str1.*lower*(), str1.*capitalize*(), str1.*title*(), str1.*swapcase*() thì chuỗi *str1* vẫn không thay đổi.

3. Phép toán in

Phép toán: *str1* **in** *st* trả về *True* nếu *str1* là chuỗi con của *st*; ngược lại, phép toán trả về giá trị *False*.

Ví dụ 9. Chương trình:

```
str1 = "ab"
st = "ab12ab"
if str1 in st: print("La chuoi con")
else: print("Khong la chuoi con")
```

☞ Kết quả:

```
La chuoi con
```

4. Phương thức đếm số lần xuất hiện

➧ Phương thức <*biến chuỗi*>.**count**(*str1*)

Trả về số lần xuất hiện của chuỗi *str1* trong <*biến chuỗi*>.

Ví dụ 10. Chương trình:

```
str1 = "ab"
st = "ab12ab"
n = st.count(str1)
print("So lan xuat hien: ", n)
```

☞ Kết quả:

```
So lan xuat hien:  2
```

❖ **Tham số với phương thức** `count()`

- Phương thức <*biến chuỗi*>.**count**(*str*1, *i*) trả về số lần xuất hiện của chuỗi *str*1 trong <*biến chuỗi*> tính từ chỉ số *i*.
- Phương thức <*biến chuỗi*>.**count**(*str*1, *i*, *j*)

Trả về số lần xuất hiện của chuỗi *str*1 trong <*biến chuỗi*> tính từ chỉ số *i* đến chỉ số *j* – 1.

Ví dụ 11. Chương trình:

```
str1 = "ab"
st = "ab12abab"
n = st.count(str1, 1)
m = st.count(str1, 1, 6)
print("So lan xuat hien: ",n)
print("So lan xuat hien: ",m)
```

☞ Kết quả:

```
So lan xuat hien:  2
So lan xuat hien:  1
```

5. Phương thức tìm kiếm

- Phương thức <*biến chuỗi*>.**find**(*str*1)

Trả về vị trí chỉ số đầu tiên mà chuỗi *str*1 xuất hiện trong <*biến chuỗi*>. Nếu *str*1 không xuất hiện trong <*biến chuỗi*> thì kết quả trả về là -1.

Ví dụ 12. Chương trình:

```
str1 = "ab"
str2 = "abc"
st   = "acabab"
print("Vi tri dau tien str1 trong st: ",st.find(str1))
print("Vi tri dau tien str2 trong st: ",st.find(str2))
```

☞ Kết quả:

```
Vi tri dau tien str1 trong st:  2
Vi tri dau tien str2 trong st:  -1
```

❖ **Tham số với phương thức** `find()`

- Phương thức *<biến chuỗi>*.**find**(*str*1, *i*) trả vị trí chỉ số đầu tiên mà chuỗi *str*1 xuất hiện trong *<biến chuỗi>* tính từ chỉ số *i*.
- Phương thức *<biến chuỗi>*.**find**(*str*1, *i*, *j*) trả vị trí chỉ số đầu tiên mà chuỗi *str*1 xuất hiện trong *<biến chuỗi>* tính từ chỉ số *i* đến chỉ số *j* – 1.

Ví dụ 13. Chương trình:

```
str1 = "ab"
st   = "abababab"
print("Vi tri dau tien: ",st.find(str1,1))
print("Vi tri dau tien: ",st.find(str1,1,3))
```

☞ Kết quả:

```
Vi tri dau tien:  2
Vi tri dau tien:  -1
```

❖ ***Chú ý***

Phương thức **st.index(str1)** *cũng trả về vị trí chỉ số đầu tiên của chuỗi* **str1** *xuất hiện trong* **st**, *nhưng khi* **str1** *không xuất hiện trong* **st** *thì chương trình sẽ báo lỗi.*

6. Phương thức thay thế

- Phương thức <*biến chuỗi*>.**replace**(*str_old, str_new*)

Trả về chuỗi được tạo thành bằng cách thay thế tất cả các chuỗi con *str_old* bằng chuỗi *str_new* trong <*biến chuỗi*>.

Ví dụ 14. Chương trình:

```
st   = "aaabaa"
str1 = st.replace("aa","AA")
print(st)
print(str1)
```

☞ Kết quả:

```
aaabaa
AAabAA
```

❖ **Tham số với phương thức** `replace()`

- Phương thức <*biến chuỗi*>.**replace**(*str_old, str_new, num_max*)

Trả về một chuỗi được tạo thành bằng cách thay thế tất cả các chuỗi con *str_old* bằng chuỗi *str_new* nhưng số lần thay thế nhiều nhất là *num_max*. Việc thay thế sẽ thực hiện lần lượt đối với các chuỗi con từ trái sang phải.

Ví dụ 15. Chương trình:

```
st = "aaaaaabb"
str1 = st.replace("aa","AA",2)
print(str1)
```

☞ Kết quả:

```
AAAAaabb
```

7. Phương thức xóa các kí tự ở bên trái, bên phải chuỗi

- Phương thức <*biến chuỗi*>.**lstrip()**

Trả về một chuỗi được tạo thành bằng cách xóa tất cả các kí tự trắng (white space) ở bên trái <*biến chuỗi*>.

Ví dụ 16. Chương trình:

```
st = " aaAbb"
str1 = st.strip()
print(str1)
```

☞ Kết quả:

```
aaAbb
```

- Phương thức <***biến chuỗi***>.**rstrip()**

Trả về một chuỗi được tạo thành bằng cách xóa tất cả các kí tự trắng ở bên phải <*biến chuỗi*>.

Ví dụ 17. Chương trình:

```
st = "aaAbb    "
str1 = st.strip()
print(str1)
```

☞ Kết quả:

```
aaAbb
```

- Phương thức <***biến chuỗi***>.**strip()**

Trả về một chuỗi được tạo thành bằng cách xóa tất cả các kí tự trắng ở cả bên trái và bên phải <*biến chuỗi*>.

Ví dụ 18. Chương trình:

```
st = "   aaAbb    "
str1 = st.strip()
print(str1)
```

☞ Kết quả:

```
aaAbb
```

❖ Tham số với phương thức `lstrip(), rstrip(), strip()`

- Phương thức <***biến chuỗi***>.**lstrip**([*Các kí tự*])

Trả về chuỗi được tạo thành bằng cách xóa các kí tự thuộc tập [*Các kí tự*] nằm liền kề bên trái <*biến chuỗi*>.

- Phương thức <***biến chuỗi***>.**rstrip**([*Các kí tự*])

Trả về chuỗi được tạo thành bằng cách xóa các kí tự thuộc tập [*Các kí tự*] nằm liền kề bên phải <*biến chuỗi*>.

- Phương thức <***biến chuỗi***>.**strip**([*Các kí tự*])

Trả về chuỗi được tạo thành bằng cách xóa các kí tự thuộc tập [*Các kí tự*] nằm liền kề bên trái hoặc bên phải <*biến chuỗi*>.

Ví dụ 19. Chương trình:

```
st = "aaACCbab"
stL = st.lstrip("aAb")
stR = st.rstrip("aAb")
str1 = st.strip("aAb")
print(stL)
print(stR)
print(str1)
```

Kết quả:

```
CCbab
aaACC
CC
```

8. Căn chỉnh chuỗi

- Phương thức <***biến chuỗi***>.**ljust**(*width*)

Trả về một chuỗi có độ dài *width* bằng cách thêm các kí tự trắng về bên phải <*biến chuỗi*>. Nói cách khác, phương thức tạo ra một chuỗi độ dài *width*, trong đó <*biến chuỗi*> được căn bên trái, các kí tự trắng được thêm về bên phải.

Ví dụ 20. Chương trình:

```
st   = "abcd"
str1 = st.ljust(8)
str2 = str1 + "abcd"
print(str2)
```

☞ Kết quả:

```
abcd     abcd
```

➧ Phương thức <***biến chuỗi***>.**rjust**(*width*)

Trả về một chuỗi có độ dài *width* bằng cách thêm các kí tự trắng về bên trái <*biến chuỗi*>.

Ví dụ 21. Chương trình:

```
st   = "abcd"
str1 = st.rjust(8)
str2 = "--------"
print(str1)
print(str2)
```

☞ Kết quả:

```
    abcd
--------
```

➧ Phương thức <***biến chuỗi***>.**center**(*width*)

Trả về chuỗi có độ dài *width* bằng cách thêm các kí tự trắng vào cả hai bên (trái, phải) sao cho <*biến chuỗi*> được căn ở chính giữa.

Ví dụ 22. Chương trình:

```
st   = "abcd"
str1 = st.center(8)
str2 = "--------"
print(str1)
print(str2)
```

☞ Kết quả:

```
  abcd
--------
```

❖ **Tham số với phương thức** `ljust(), rjust(), center()`

Ở trên, các phương thức **`ljust(), rjust(), center()`** sử dụng tham số mặc định là kí tự trắng. Các phương thức này có thể sử dụng tham số là các kí tự khác kí tự trắng.

*<**biến chuỗi**>*.**`ljust`**(*width, [Kí tự thêm vào]*)

*<**biến chuỗi**>*.**`rjust`**(*width, [Kí tự thêm vào]*)

*<**biến chuỗi**>*.**`center`**(*width, [Kí tự thêm vào]*)

Ví dụ 23. Chương trình:

```
st   = "abcd"
strL = st.ljust(8,"-")
strR = st.rjust(8,"+")
strC = st.center(8,"#")
print(strL)
print(strR)
print(strC)
```

☞ Kết quả:

```
abcd----
++++abcd
##abcd##
```

❖ ***Chú ý***

Python còn cung cấp một phương thức thêm các kí tự '0' vào bê trái. Điều này hay gặp khi thực hiện với các chuỗi là các kí tự số.

*Phương thức <**biến chuỗi**>.**`zfill`**(width)*

*Trả về chuỗi có độ dài **width** bằng cách thêm vào các kí tự ch số 0 vào bên trái <biến chuỗi>.*

Ví dụ 24. Chương trình:

```
st   = "1234"
str1 = st.zfill(10)
print(str1)
```

☞ Kết quả:

```
00001234
```

9. Một số phương thức is___

- Phương thức <*biến chuỗi*>.**isdigit()**

Trả về giá trị *True* nếu <*biến chuỗi*> chỉ chứa các kí tự chữ số, ngược lại trả về giá trị *False*.

Ví dụ 25. Chương trình:

```
str1 = "1234"
str2 = "123a"
print(str1.isdigit())
print(str2.isdigit())
```

☞ Kết quả:

```
True
False
```

- Phương thức <*biến chuỗi*>.**isspace()**

Trả về giá trị *True* nếu <*biến chuỗi*> chỉ chứa các kí tự trắng, ngược lại trả về giá trị *False*.

Ví dụ 26. Chương trình:

```
str1 = "   "
str2 = "123a "
print(str1.isspace())
print(str2.isspace())
```

☞ Kết quả:

```
True
False
```

- Phương thức <*biến chuỗi*>.**islower()**

Trả về giá trị *True* nếu <*biến chuỗi*> chỉ chứa các kí tự chữ thường, ngược lại trả về giá trị *False*.

Ví dụ 27. Chương trình:

```
str1 = "1234ab"
str2 = "123aB"
print(str1.islower())
print(str2.islower())
```

☞ Kết quả:

```
True
False
```

☙ Phương thức <*biến chuỗi*>.**isupper()**

Trả về giá trị *True* nếu <*biến chuỗi*> chỉ chứa các kí tự chữ hoa, ngược lại trả về giá trị *False*.

Ví dụ 28. Chương trình:

```
str1 = "1234AB"
str2 = "123aB"
print(str1.isupper())
print(str2.isupper())
```

☞ Kết quả:

```
True
False
```

☙ Phương thức <*biến chuỗi*>.**isalpha()**

Trả về giá trị *True* nếu <*biến chuỗi*> chỉ chứa các kí tự thuộc 'a', 'b',.., 'z'; 'A', 'B', .. 'Z', ngược lại trả về giá trị *False*.

Ví dụ 29. Chương trình:

```
str1  = "abcABC"
str2  = "aBC123"
print(str1.isalpha())
print(str2.isalpha())
```

☞ Kết quả:

```
True
False
```

- Phương thức <*biến chuỗi*>.**istitle()**

Trả về giá trị *True* nếu <*biến chuỗi*> có các từ đều chữ cái đầu là chữ hoa, ngược lại trả về giá trị *False*.

Ví dụ 30. Chương trình:

```
str1 = "Tu Hoc Python"
str2 = "tu Hoc python"
print(str1.istitle())
print(str2.istitle())
```

☞ Kết quả:

```
True
False
```

10. Hàm max, min trên chuỗi

- Hàm **max**(<*biến chuỗi*>)

Trả về kí tự lớn nhất trong <*biến chuỗi*>.

- Hàm **min**(<*biến chuỗi*>)

Trả về kí tự nhỏ nhất trong <*biến chuỗi*>.

Ví dụ 31. Chương trình:

```
str1 = "abcd"
print(min(str1))
print(max(str1))
```

☞ Kết quả:

```
a
d
```

. Hàm eval() cho biểu thức toán học

Python cung cấp một hàm rất mạnh cho phép tính giá trị của các ểu thức toán học đúng. Đó là hàm **eval()**.

Hàm **eval(*str1*)** trả về giá trị của biểu thức toán học đúng ược viết dưới dạng chuỗi *str1*.

Ví dụ 32. Chương trình:

```
str1 = "1+2+3"
print(eval(str1))
a = 10
b = 20
str2 = "1 + 2*(a+b)"
print(eval(str2))
```

☛ Kết quả:

```
6
61
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Từ trong chuỗi

Cho chuỗi str1. Ta gọi một từ trong chuỗi là dãy các kí tự liên tiếp (khác kí tự trắng và dấu phẩy) và được phân cách bởi các kí tự trắng hoặc dấu phẩy. Ví dụ: str1 = "Tu hoc, lap, trinh Python", gồm năm từ "Tu", "hoc", "lap", "trình", "Python". *Đưa ra danh sách các từ trong chuỗi, mỗi từ ghi trên một dòng.*

Bài tập 2. Tổng các chữ số trong chuỗi

Cho chuỗi str1 gồm các kí tự chữ số và kí tự chữ cái La-tinh. Tính tổng các chữ số trong chuỗi str1. Ví dụ: Str1 = "123abc4", tổng các chữ số bằng 1 + 2 + 3 + 4 = 10.

Bài tập 3. Chuỗi hoán vị

Hai chuỗi str1, str2 có cùng độ dài được gọi là hai chuỗi hoán vị của nhau nếu có thể đổi chỗ các kí tự trong chuỗi str1 để trở thành

chuỗi str2. Cho hai chuỗi str1, str2, kiểm tra xem có phải là hai chuỗi hoán vị của nhau không?

Bài tập 4. Đặt phép toán

Cho chuỗi kí tự có dạng A = B. Trong đó A và B là hai chuỗi chỉ chứa các kí tự chữ số, tức là A, B là hai số nguyên. Hãy chèn một kí tự '+' vào giữa các chữ số của A và một kí tự '+' và giữa các kí tự của B để được một biểu thức toán học đúng có dạng: C + D = E + F và đưa biểu thức này ra màn hình.

Ví dụ: Biểu thức A = B có dạng: 111 = 120.

Ta có thể đặt các kí tự '+' để được biểu thức toán học đúng là: 11 + 1 = 12 + 0.

Nhận thấy, có thể không có cách đặt kí tự '+' để được biểu thức toán học đúng trong nhiều biểu thức dạng A = B. Với trường hợp này, thông báo "Khong co cach dat".

Bài tập 5. Mật mã Caesar

Trong mật mã học, mật mã Caesar, còn gọi là mật mã dịch chuyển, là một trong những mật mã đơn giản và được biết đến nhiều nhất. Mật mã Caesar là một dạng của mật mã thay thế, trong đó mỗi ký tự trong văn bản được thay thế bằng một ký tự cách nó một đoạn không đổi d trong bảng chữ cái để tạo thành văn bản mã hóa. Ví dụ, Nếu xét bảng chữ cái để viết văn bản là: 'A', 'B', 'C', .. 'Z', độ dịch chuyển là d = 2. Khi đó kí tự 'A' được mã hóa thành kí tự 'C'; kí tự 'B' được mã hóa thành 'D', .., kí tự 'X' mã hóa thành 'Z', kí tự 'Y' mã hóa thành 'A', kí tự 'Z' mã hóa thành 'B'. Giá trị d được gọi là khóa, khi biết khóa d, thì từ văn bản mã hóa, ta có thể tìm ra được văn bản gốc ban đầu bằng cách dịch chuyển theo hướng ngược lại đến kí tự cách một đoạn d.

Phương pháp được đặt tên theo Caesar - vị hoàng đế đã sử dụng nó thường xuyên trong công việc.

Yêu cầu: Cho văn bản dưới dạng một chuỗi *str1* gồm các kí tự thuộc 'A', 'B', 'C', .. 'Z'. *Hãy đưa ra chuỗi mã hóa theo mã Caesar với độ dịch chuyển là d* ($0 < d < 26$). Ví dụ: *str1* = "HELLOWOLRD", $d = 2$, ta có văn bản mã hóa là: "JGNNQYQNTF".

. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Từ trong chuỗi

Ý tưởng thuật toán

Sử dụng câu lệch tách **str1.split(' ')**để tách chuỗi **str1** ành những chuỗi con không chứa kí tự trống.

Với mỗi chuỗi con vừa được tách được, ta lại sử dụng lệnh tách **plit(',')** để tách thành các chuỗi con nhỏ hơn không chứa kí tự dấu hẩy. Các chuỗi con cuối cùng nhận được là các từ trong chuỗi ban đầu.

Chương trình

```
str1 = "Tu hoc, lap,trinh Python"
#list1 la danh sach cac chuoi con khong chua ki
tu trong
list1 = str1.split(' ')
list2 = []
for x in list1:
   list2 = list2 + x.split(',')
print("Cac tu trong chuoi:")
for x in list2:
   if x != '':
      print(x)
```

Bài tập 2. Tổng các chữ số trong chuỗi

Ý tưởng thuật toán

Duyệt qua các kí tự trong chuỗi, nếu là kí tự chữ số thì chuyển kí tự đó thành kí tự số và cộng kí tự số này vào tổng.

Để kiểm tra một kí tự x là kí tự số ta có thể thực hiện:

'0' <= x and x <= '9' hoặc **x.isdigit()**

Chương trình

```
str1 = "123bc127dd"
s = 0
for x in str1:
   if x.isdigit(): #co the dung '0' <= x and x
<= '9'
      s = s + int(x)
print(s)
```

Bài tập 3. Chuỗi hoán vị

Ý tưởng thuật toán

Sắp xếp các kí tự trong chuỗi str1 theo thứ tự tăng dần.

Sắp xếp các kí tự trong chuỗi str2 theo thứ tự tăng dần.

Nếu hai chuỗi sau khi sắp xếp bằng nhau thì hai chuỗi ban đầu là hai chuỗi hoán vị, ngược lại, hai chuỗi ban đầu là hai chuỗi không phải hoán vị của nhau.

Chương trình

```
str1 = "abcc"
str2 = "ccba"
str1 = ''.join(sorted(str1))
str2 = ''.join(sorted(str2))
if str1 == str2:
   print("Hai chuoi la hoan vi cua nhau")
else:
   print("Hai chuoi khong la hoan vi cua nhau")
```

❖ ***Chú ý***

- *Hàm* **sorted**(<str>)

Sắp xếp các kí tự trong chuỗi <str> theo thứ tự tăng dần, nhưng kết quả trả về là một danh sách.

- *Phương thức <str>.***join**(<tr1>)

Trả về một chuỗi bằng cách ghép chuỗi <str> với các phần tử của <str1>. <str1> có thể là một chuỗi, có thể là một danh sách gồm các phần tử là chuỗi, hay là một tập hợp các phần tử của nó là các chuỗi.

Bài tập 4. Đặt phép toán

Ý tưởng thuật toán

Xét hai chuỗi gồm các kí tự chữ số A và B.

Đặt $n = len(A)$, như vậy chuỗi A gồm các kí tự: $A[0], A[1], .., A[n-1]$.

Đặt $m = len(B)$, như vậy chuỗi B gồm các kí tự $B[0], B[1], .., B[m-1]$.

Thực hiện duyệt tất cả các vị trí đặt dấu cộng trong chuỗi A và B tương ứng với duyệt tất cả các cặp chỉ số (i, j):

Đặt phép toán '+' kề trước vị trí $A[i]$ (với $i = 1, 2, .., n - 1$) trong chuỗi A.

Đặt phép toán '+' kề trước vị trí $B[j]$ (với $j = 1, 2, .., m - 1$) trong chuỗi B.

Khi đó:

Chuỗi A được chia thành hai chuỗi: $aL = A[:i]$ và $aR = A[i:]$.

Chuỗi B được chia thành hai chuỗi: $bL = B[:j]$ và $bR = B[j]$.

Nếu $int(aL) + int(aR) == int(bL) + int(bR)$ thì ta có một cách đặt dấu phép toán '+' thỏa mãn.

Chương trình

```
str = "303=600"
i = str.find('=')
A = str[:i]
B = str[(i+1):]
n = len(A)
m = len(B)
dem = 0                    #dem so cach dat
for i in range(1,n):
   for j in range(1,m):
      aL = A[:i]
      aR = A[i:]
      bL = B[:j]
      bR = B[j:]
      if int(aL) + int(aR) == int(bL) + int(bR):
         dem = dem + 1
         print(aL,'+',aR,'=',bL,'+',bR)
if dem == 0:
   print("Khong co canh dat")
```

Bài tập 5. Mật mã Caesar

Ý tưởng thuật toán

Tạo một chuỗi mẫu:

STR = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ" được tạo bởi 2 lần chuỗi gồm các kí tự từ 'A' đến 'Z'.

Khi đó với mỗi kí tự $STR[i]$ sẽ được mã hóa thành kí tự $STR[i + d]$ với $(i = 0, 1, .., 25)$.

Mã hóa văn bản:

Xét lần lượt từng kí tự trong văn bản cần mã hóa.

Với một kí tự ch, ta tính $i = STR.find(ch)$, nghĩa là $STR[i] == ch$ Do vậy kí tự ch sẽ được mã hóa thành kí tự $STR[i + d]$.

Chương trình

```
STR = ""
for i in range(26):
   STR = STR + chr(ord('A')+i)
STR = STR*2
d = 2
chuoi_goc = "HELLOWOLRD"
chuoi_ma = ""
for x in chuoi_goc:
   i = STR.find(x)
   chuoi_ma = chuoi_ma + STR[i+d]
print("Chuoi duoc ma la:")
print(chuoi_ma)
```

❖ ***Chú ý***

Để tạo xâu mẫu STR như trên, ta có thể gõ trực tiếp vào chương trình nguồn. Ngoài ra, ta có thể sử dụng hai hàm sau và các câu lệnh như trong chương trình.

- *Hàm* **ord**(*ch*)

Trả về mã của kí tự ch trong bảng mã ASCII.

- *Hàm* **chr**(*h*)

Trả về kí tự trong bảng mã ASCII có mã bằng h.

CHỦ ĐỀ 10. KIỂU DỮ LIỆU DANH SÁCH - LIST

A. KIẾN THỨC CƠ BẢN

Trong phần này, ta sẽ làm quen với một trong những kiểu dữ liệu cực kì mạnh của Python đó là danh sách (*list*).

1. Khai báo kiểu dữ liệu danh sách

Danh sách là một tập hợp các phần tử. Ví dụ, danh sách chữ cái, màu sắc, môn học... Để khai báo một danh sách, ta sử dụng cú pháp như sau:

<biến danh sách> = [*phần tử 0*, *phần tử 1*, ..., *phần tử* $n-1$]

Các phần tử trong danh sách được liệt kê trong cặp dấu ngoặc vuông [] và phân cách nhau bởi dấu phẩy.

Ví dụ 1. Một số khai báo:

```
list1 = []
```

☛ khai báo một danh sách rỗng.

```
list2 = ['Toan', 'Ly', 'Hoa', 'Sinh', 'Tin']
```

☛ khai báo một danh sách gồm 5 phần tử.

```
list3 = [1, 2, 3, 4, 5, 'Toan', 'Ly', 'Hoa']
```

☛ khai báo một danh sách gồm 8 phần tử.

Các phần tử trong danh sách được đánh chỉ số bắt đầu từ chỉ số 0, 1, ... (từ trái sang phải) và $-1, -2, ..$ (từ phải sang trái). Cách đánh chỉ số này giống như trong đánh chỉ số các kí tự trong kiểu dữ liệu chuỗi.

Ví dụ 2. Chương trình:

```
list1 = ['Toan', 'Ly', 'Hoa', 'Sinh', 'Tin']
print("list1   : ",list1)
print("list1[0]: ",list1[0])
print("list1[1]: ",list1[1])
print("list1[-1]: ",list1[-1])
print("list1[-2]: ",list1[-2])
```

☞ Kết quả:

```
list1   :  ['Toan', 'Ly', 'Hoa', 'Sinh', 'Tin']
list1[0]:  Toan
list1[1]:  Ly
list1[-1]:  Tin
list1[-2]:  Sinh
```

2. Lấy số phần tử trong danh sách

Hàm ***Len****(<biến danh sách>)*

Trả về số lượng phần tử trong *<biến danh sách>*.

Ví dụ 3. Chương trình:

```
list1 = ['Toan', 'Ly', 'Hoa', 'Sinh', 'Tin']
print("So phan tu: ",len(list1))
```

☞ Kết quả:

```
So phan tu:  5
```

3. Truy cập đến các phần tử trong danh sách

<biến danh sách> [*chỉ số*]

Truy cập tới phần tử có [*chỉ số*] trong *<biến danh sách>*.

Ví dụ 4. Chương trình:

```
list1 = [1, 2, 3]
n = len(list1)
for i in range(n):
   print(list1[i])
```

☞ Kết quả:

```
1
2
3
```

❖ *Chú ý*

- *Nếu truy cập vào chỉ số không tồn tại trong danh sách thì trình thông dịch sẽ báo lỗi* **IndexError: list index out of range.**
- *Để đưa ra tất cả các phần tử của danh sách, ta có thể thực hiện*:

```
list1 = [1, 2, 3]
for x in list1:
    print(x)
```

- Các phần tử trong danh sách có thể thay đổi được bằng cách:

<biến danh sách>[chỉ số] = <giá trị mới>.

4. Lấy các phần tử liên tiếp trong danh sách

- Phương thức *<biến danh sách>* [*a*: *b*]

Trả về danh sách gồm các phần tử từ chỉ số *a* đến chỉ số *b* – 1 trong *<biến danh sách>*.

Ví dụ 5. Chương trình:

```
list1 = ['a', 'b', 'c', 'd','e']
list2 = list1[1:4]
print(list2)
```

☛ Kết quả:

```
['b', 'c', 'd']
```

- Phương thức *<biến danh sách>*[: *b*]

Trả về danh sách gồm các phần tử từ chỉ số 0 đến chỉ số *b* – 1.

Như vậy: <biến danh sách>[: *b*] = = <biến danh sách>[0: *b*].

- Phương thức *<biến danh sách>*[*a*:]

Trả về danh sách gồm các phần tử từ chỉ số *a* đến hết danh sách.

Như vậy: <biến danh sách>[*a*:] = = <biến danh sách>[*a*: *n*], trong đó *n* = **len**(<biến danh sách>).

- Phương thức <*biến danh sách*>[*a*: *b*: *c*]

Trả về danh sách gồm tất cả các phần tử có chỉ số: $[a]$, $[a + c]$, $[a + 2c]$, .. , $[a + tc]$ với $a + tc < b$, trong <*biến danh sách*>.

Ví dụ 6. Chương trình:

```
list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list2 = list1[:4]
list3 = list1[4:]
list4 = list1[0:8:3]
print(list2)
print(list3)
print(list4)
```

☞ Kết quả:

```
[0, 1, 2, 3]
[4, 5, 6, 7, 8, 9, 10]
[0, 3, 6]
```

5. Nối (ghép) danh sách và lặp danh sách

- Phép nối (ghép) danh sách:

<*danh sách* 1> + <*danh sách* 2>

Trả về một danh sách gồm các phần tử của danh sách <*danh sách* 1> và <*danh sách 2*> được ghép liên tiếp nhau.

<*danh sách* 1> + <*danh sách* 2> + .. + <*danh sách n*>

Trả về một danh sách gồm các phần tử của danh sách <*danh sách* 1> và <*danh sách* 2>, .., <*danh sách n*> được ghép liên tiếp nhau.

Ví dụ 7. Chương trình:

```
list1 = [1, 2, 3, 4]
list2 = [2, 3, 4, 5]
list3 = list1 + list2
print(list3)
```

☞ Kết quả:

```
[1, 2, 3, 4, 2, 3, 4, 5]
```

- Phép lặp danh sách:

*<danh sách>***<số lần lặp>*

*<số lần lặp>***<danh sách>*

Trả về danh sách bằng cách ghép *<danh sách>* với số lần *<số lần lặp>*.

Ví dụ 8. Chương trình:

```
list = [1, 2, 3]
list2 = list*3
list3 = 3*list
print(list2)
print(list3)
```

☞ Kết quả:

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

6. Thêm phần tử vào danh sách, mở rộng danh sách

- Phương thức *<biến danh sách>*.**append(x)**

Thêm một phần tử x vào cuối *<biến danh sách>*.

Ví dụ 9. Chương trình:

```
list1 = [1, 2, 3, 4]
list1.append(5)
print(list1)
```

☞ Kết quả:

```
[1, 2, 3, 4, 5]
```

- Phương thức *<biến danh sách>*.**extend**(*list2*)

Nối danh sách *list2* vào <biến danh sách>.

Ví dụ 10. Chương trình:

```
list1 = [1, 2, 3, 4]
list2 = [2, 3]
list1.extend(list2)
print(list1)
```

☞ Kết quả:

```
[1, 2, 3, 4, 2, 3]
```

Ví dụ 11. Chương trình:

```
list1 = [1, 2, 3, 4]
list1.extend(range(4))
print(list1)
```

☞ Kết quả:

```
[1, 2, 3, 4, 0, 1, 2, 3]
```

❖ ***Chú ý***

Ta có hai câu lệnh: `list1.extend(list2)` và `list1 = list1 + list2` *là giống nhau.*

Có sự khác nhau giữa hai phương thức **append()** *và* **extend()**.

Ví dụ 12. Chương trình:

```
list1 = [1, 2, 3, 4]
list2 = [1, 2, 3, 4]
list3 = [2, 3]
list1.extend(list3)
list2.append(list3)
print(list1)
print(list2)
```

☞ Kết quả:

```
[1, 2, 3, 4, 2, 3]
[1, 2, 3, 4, [2, 3]]
```

Như vậy, **list1.append**(<*đối tượng*>), thêm <*đối tượng*> v
list1 như một phần tử của **list1**, **list1.extend**(<*đối tượng*
thêm các phần tử của <*đối tượng*> vào **list1**.

7. Phép toán in và not in

Phép toán: <*phần tử*> **in** <*danh sách*>

Trả về giá trị ***True*** nếu <*phần tử*> thuộc <*danh sách*>, ngược
trả lại giá trị ***False***.

Ví dụ 13. Chương trình:

```
list1 = [1, 2, 3, 4]
x = 1
y = 5
if x in list1: print("x thuoc list1")
if y in list1: print("y thuoc list1")
else: print("y khong thuoc list1")
```

☛ Kết quả:

```
x thuoc list1
y khong thuoc list1
```

Phép toán: <*phần tử*> **not in** <*danh sách*>

Trả về giá trị ***True*** nếu <*phần tử*> không thuộc <*danh sác*
ngược lại phép toán trả về giá trị ***False***.

Ví dụ 14. Chương trình:

```
list1 = [1, 2, 3, 4]
x = 5
if x not in list1: print("x khong thuoc list1"
else: print("x thuoc list1")
```

☛ Kết quả:

```
x khong thuoc list1
```

8. Phương thức `index()`

Phương thức <*biến danh sách*>.**index**(x) trả về chỉ số của phần tử đầu tiên trong <*biến danh sách*> bằng x.

Ví dụ 15. Chương trình:

```
list1 = [1, 2, 3, 2]
print("Chi so phan tu dau tien bang 2 la: ",
list1.index(2))
```

☞ Kết quả:

```
Chi so phan tu dau tien bang 2 la:  1
```

❖ ***Chú ý***

*Khi x không thuộc list*1 *và sử dụng phương thức list*1.**index(x)** *thì trình thông dịch sẽ báo lỗi* **unexpected indent**. *Do vậy, ta nên kết hợp sử dụng phép toán* **in** *và phương thức* **index()**.

Ví dụ 16. Chương trình:

```
list1 = [1, 2, 3, 2]
x = 5
if x in list1:
  print("Chi so phan tu dau tien bang x la:
",list1.index(x))
else:
  print("x khong thuoc list1")
```

☞ Kết quả:

```
x khong thuoc list1
```

Dạng tham số của phương thức `index()`

Phương thức <*biến danh sách*>.**index**(x, i) trả về chỉ số của phần tử đầu tiên trong các phần tử có chỉ số từ i thuộc <*biến danh sách*> bằng x.

Ví dụ 17. Chương trình:

```
list1 = [1, 1, 2, 3, 2, 1]
x = 1
print("Chi so: ",list1.index(x))
print("Chi so: ",list1.index(x,1))
```

☛ Kết quả:

```
Chi so:  0
Chi so:  1
```

9. Phép toán min(), max() trên danh sách

Phép toán **min(**<*biến danh sách*>**)** trả về giá trị của phần tử nhỏ nhất trong <*biến danh sách*>.

Phép toán **max(**<*biến danh sách*>**)** trả về giá trị của phần tử lớn nhất trong <*biến danh sách*>.

Ví dụ 18. Chương trình:

```
list1 = [1, 1, 2, 3, 5]
print("Gia tri nho nhat: ",min(list1))
print("Gia tri lon nhat: ",max(list1))
```

☛ Kết quả:

```
Gia tri nho nhat:  1
Gia tri lon nhat:  5
```

10. Hàm sum() trên danh sách

Cho *list1* là danh sách các số. Để tính tổng các phần tử trong *list1*, ta sử dụng **sum(***list1***)**.

Ví dụ 19, chương trình:

```
list1 = [1, 2, 3, 4]
s = sum(list1)
print("Tong cac phan tu: ",s)
```

☛ Kết quả:

```
Tong cac phan tu:  10
```

❖ ***Nhận xét***

Kiểu dữ liệu danh sách (list) trong Python là một dạng mở rộng kiểu dữ liệu mảng (array). Kiểu dữ liệu mảng trong các ngôn ngữ lập trình khác (như C++, Java, ..) là danh sách các phần tử cùng kiểu. Python cơ bản không cung cấp kiểu dữ liệu mảng như vậy mà thay vào đó là kiểu dữ liệu list. Các phần tử trong kiểu dữ liệu list có thể giống nhau, cũng có thể khác nhau. Các phiên bản Python 3.x có cung cấp một mô-đun ***array*** *cho phép người lập trình sử dụng kiểu dữ liệu này như các ngôn ngữ lập trình khác.*

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tổng các số nguyên lẻ

Nhập số nguyên dương n và danh sách gồm n số nguyên $a_0, a_1, .., a_{n-1}$. *Hãy tính tổng các số nguyên lẻ trong dãy.*

Bài tập 2. Tổng *k* số hạng liên tiếp có tổng lớn nhất.

Nhập số nguyên dương n , k ($k \leq n$) *và danh sách gồm n số nguyên* $a_0, a_2, .., a_{n-1}$. *Tìm k phần tử liên tiếp trong dãy:* a_i, a_{i+1}, .., a_{i+k-1} *sao cho tổng* $a_i + a_{i+1} + .. + a_{i+k-1}$ *đạt giá trị lớn nhất. Ví dụ, dãy gồm 6 số hạng: 1, 2, -19, 3, 4, -1; k = 3. Tổng 3 số hạng liên tiếp có tổng lớn nhất là: 3, 4, -1, tổng bằng 3 + 4 + (-1) = 6.*

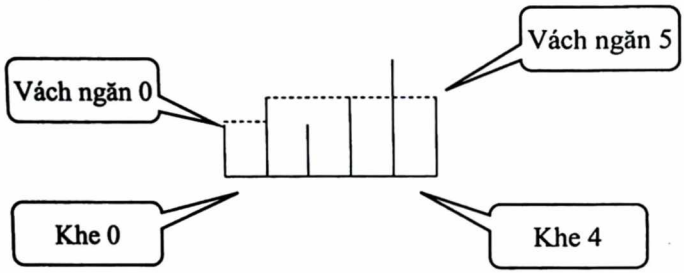
Bài tập 3. Sàng số nguyên tố Eratosthenes

Nhập số nguyên dương n ($2 \leq n \leq 10^7$). *Hãy đưa ra danh sách các số nguyên tố nhỏ hơn hoặc bằng n. Ví dụ,* $n = 10$, *danh sách các số nguyên tố là:* $2, 3, 5, 7$.

Bài tập 4. Khe ngăn nước

Một bình chứa nước được thiết kế rất đặc biệt. Nó gồm $n + 1$ *vách ngăn, tạo ra n khe nhỏ. Các vách ngăn được đánh số thứ tự từ 0 đến n (theo hướng từ trái sang phải). Vách ngăn thứ i có độ cao là* h_i, *các vách ngăn cách nhau 1 (đơn vị độ dài). Đáy mỗi vách ngăn là một hình vuông độ dài 1. Khi nước ở vách ngăn có độ cao h thì ta xem lượng nước chứa trong đó là h (đơn vị khối). Người ta đổ đầy nước vào bình cho đến khi cứ đổ vào là nước tràn ra. Hãy tính lượng nước chứa được trong bình.*

Ví dụ:



Với n = 5, độ cao các vách ngăn $h_0 = 2$, $h_1 = 3$, $h_2 = 2$, $h_3 = 3$ $h_4 = 6$, $h_5 = 3$ (hình vẽ). Khi đó lượng nước chứa được nhiều nhất ứng với mức nước trên đường gạch chéo. Lượng nước ở khe 0 chứa bằng 2 khe 1 bằng 3, khe 2 bằng 3, khe 3 bằng 3, khe 4 bằng 3. Như vậy lượng nước nhiều nhất bằng 2 + 3 + 3 + 3 + 3 = 14 (đơn vị khối).

Bài tập 5. Chọn các số

Cho dãy số nguyên gồm n số hạng $a_0, a_2, .. , a_{n-1}$. Hãy chọn các số hạng trong dãy sao cho thỏa mãn:

- Không chọn hai số hạng kề nhau, tức là không chọn hai số hạng a_i và a_{i+1}.

- Tổng các số hạng được chọn có giá trị lớn nhất.

Ví dụ: Dãy: 1, -2, 3, 2, 6. Ta chọn các số hạng 1, 3, 6.

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tổng các số nguyên lẻ

Ý tưởng thuật toán

- Cho biến lưu giá trị tổng s = 0.

- Duyệt lần lượt các số hạng trong dãy, ứng với mỗi số hạng a_i lẻ ta cộng a_i vào tổng s.

Chương trình

```
n = int(input("Nhap so phan tu cua day "))
list = []
```

```
for i in range(n):
    x = int(input())
    list.append(x)
s = 0
for x in list:
    if x%2 != 0:
        s = s + x
print("Tong cac so hang le: ", s)
```

Bài tập 2. Tổng k số hạng liên tiếp có tổng lớn nhất

Ý tưởng thuật toán

Xét tổng gồm k số hạng liên tiếp $a_i + a_{i+1} + .. + a_{i+k-1}$ $(0 \leq i \leq n-k)$

Thuật toán 1

Đặt $kq = a_1 + a_2 + .. + a_k$ là tổng của k số hạng đầu tiên trong dãy.

Thực hiện duyệt tất cả các giá trị của $i = 1, 3, .., n - k$.

Với mỗi giá trị của i, thực hiện tính tổng $s = a_i + a_{i+1} + .. + a_{i+k-1}$. Nếu $kq < s$ thì cập nhật $kq = s$.

Chương trình 1

```
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
list = []
for i in range(n):
    x = int(input())
    list.append(x)
s = 0
for i in range(k):
    s = s + list[i]
kq = s
```

```
for i in range(1, n - k + 1):
    s = 0
    for t in range(k):
        s = s + list[i+t]
    if kq < s:
        kq = s
print("Tong k so hang lon nhat bang: ", kq)
```

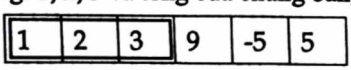
Thuật toán 2. Kỹ thuật dịch chuyển cửa sổ (*subling window*) – một thuật toán đẹp.

Ta nhận thấy, thuật toán trên sử dụng hai vòng ***for*** lồng nhau, điều này làm tăng lượng phép tính lên đáng kể khi n và k đều lớn. Cụ thể số phép tính ước tính là $(n - k + 1) \times k$. Khi $n = 10^6$, $k = 10^5$ số phép tính cần thực hiện hơn 9×10^{10}. Và cũng có nghĩa là chương trình mất thời gian đáng kể để thực hiện chương trình này. Phương pháp dịch chuyển cửa sổ cho phép giải bài toán với số phép tính thực hiện là n.

Phương pháp dịch chuyển cửa sổ:

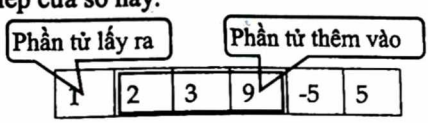
Ban đầu cửa sổ sẽ chứa k số hạng đầu tiên của dãy và ta tính tổng các số hạng trong cửa sổ này.

Ví dụ, dãy gồm $n = 6$ số hạng, $k = 3$. Cửa sổ đầu tiên gồm 3 số hạng: 1, 2, 3 và tổng của chúng bằng 6.

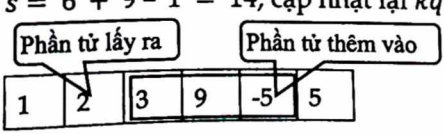


$s = 6, kq = 6$

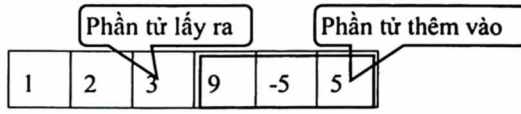
Quá trình tính toán được thực hiện bằng cách dịch chuyển liên tiếp cửa sổ này.



$s = 6 + 9 - 1 = 14$, cập nhật lại $kq = 14$



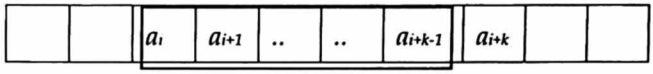
$s = 14 + (-5) - 2 = 7, kq = 14$



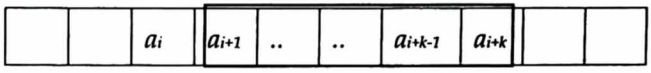
$s = 7 + 5 - 3 = 9, kq = 14$

Như vậy tổng ba số hạng liên tiếp của dãy đạt giá trị lớn nhất bằng 14.

Nhận thấy, cửa sổ luôn chứa k số hạng liên tiếp của dãy và trong quá trình dịch chuyển liên tiếp, sẽ bổ sung một số hạng vào cửa sổ và lấy ra một số hạng khỏi cửa sổ. Do vậy tổng các số hạng trong của số sau khi dịch chuyển tiếp theo sẽ bằng tổng trước đó cộng thêm phần tử mới được thêm vào và trừ đi phần tử mới được lấy ra.



$s = a_i + a_{i+1} + .. + a_{i+k-1}$



$a_{i+1} + .. + a_{i+k} = s + a_{i-k} - a_i.$

Chương trình 2

```
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
list = []
for i in range(n):
    x = int(input())
    list.append(x)
s = 0
#tong k so hang dau tien trong day
for i in range(k):
    s = s + list[i]
kq = s
#dich chuyen cua so
```

```
    for i in range(n - k):
        s = s + list[i+k] - list[i]
        if s > kq:
            kq = s
    print("Tong k so hang lon nhat bang: ", kq)
```

Bài tập 3. Sàng số nguyên tố Eratosthenes

Ý tưởng thuật toán

Sử dụng một danh sách *list* gồm N = 10^7 phần tử với ý nghĩa:

$list[i] = 1 \leftrightarrow i$ là một số nguyên tố;

$list[i] = 0 \leftrightarrow i$ không phải là một số nguyên tố.

Ban đầu, $list[0] = list[1] = 0$; $list[i] = 1$ với $i = 2,3,..,N-1$.

Ta nhận thấy với mỗi số tự nhiên a ($a = 2,3,..,N-1$), a không là một số nguyên tố khi và chỉ khi có thể phân tích $a = i \times k$ với i là một số nguyên tố và $i \leq k$. Suy ra $i \leq \sqrt{a}$.

Như vậy, ta sẽ lần lượt xét các giá trị i = 2, 3, .. , $\sqrt{N-1}$. Nếu i là số nguyên tố thì các số $a = i \times k$ ($k = i, i+1, .., \frac{N-1}{i}$) không là số nguyên tố, ta thay đổi giá trị $list[i \times k] = 0$.

Khi thực hiện tính toán xong, những giá trị i mà $list[i] = 1$ sẽ là một số nguyên tố.

Chương trình

```
N = 10000000
list = []
for i in range(N):
    list.append(1)
list[0] = list[1] = 0
canbac2_N = int(N**(1/2))
for i in range(2, 1 + canbac2_N):
```

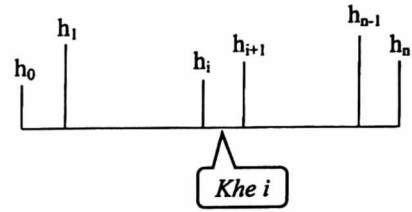
```
        if list[i] == 1:
            k = i
            while i*k < N:
                list[i*k] = 0
                k = k + 1
    print("Danh sach cac so nguyen to:")
    for i in range(2, N):
        if list[i] == 1:
            print(i, end = ' ')
```

Bài tập 4. Khe ngăn nước

Ý tưởng thuật toán

Để tính lượng nước có thể chứa được trong bình, ta sẽ tính lượng nước chứa được trong từng khe, sau đó cộng các lượng nước chứa được trong từng khe vừa tính lại với nhau.

Ta xét khe i ($i = 0, 1, .., n-1$).



Nhận thấy rằng:

Nước sẽ tràn sang bên trái và ra khỏi bình nếu mức nước tại khe i lớn hơn tất cả độ cao các vách ngăn $h_0, h_1, .. , h_i$.

Nước sẽ tràn sang bên phải và ra khỏi bình nếu mức nước tại khe i lớn hơn tất cả độ cao các vách ngăn $h_{i+1}, h_{i+2}, .., h_n$.

Như vậy, lượng nước chứa được ở khe i bằng $min(L, R)$ trong đó $L = \max(h_0, h_1, .., h_i)$, $R = \max(h_{i+1}, h_{i+2}, .. , h_n)$.

Chương trình

```
n = int (input("Nhap so khe nuoc n = "))
print("Nhap n + 1 vach ngan")
h = []
```

```
for i in range(n+1):
    x = int(input())
    h.append(x)
s = 0
for i in range(1, n+1):
    L = max(h[:i])      # L = max(h[0], .. , h[i-1])
    R = max(h[i:])      # R = max(h[i], .. , h[n])
    s = s + min(L, R)
print("Luong nuoc chua duoc trong binh: ",s)
```

❖ ***Nhận xét***

Python cung cấp các câu lệnh rất thanh lịch khi viết code, chẳng hạn để tìm giá trị lớn nhất của h[0], h[1], .., h[i - 1], *ta sử dụng* max(h[:i]); *để tìm giá trị lớn nhất của* h[i], h[i+1], .., h[n] *ta sử dụng* max(h[i:]).

Bài tập 5. Chọn các số

Ý tưởng thuật toán

Nhận thấy khi $n = 1$, ta chọn a_1.

Khi $n = 2$, ta sẽ chọn số lớn nhất trong hai số hạng a_1, a_2, tức là tổng lớn nhất bằng $\max(a_1, a_2)$.

Vậy khi n = 3, 4, .. thì sao? Rõ ràng rằng, số trường hợp phân chia sẽ nhiều hơn và không thể phân chia hết mọi trường hợp khi n lớn. Nhưng ta sẽ thấy được một điều là khi giải được bài toán cho dãy có 2 số hạng thì ta cũng dễ giải bài toán cho 3 số hạng. Khi giải được bài toán cho dãy có 3 số hạng thì cũng giải được bài toán có 4 số hạng. Và cứ tiếp tục như vậy ta sẽ giải được bài toán có n số hạng.

Kỹ thuật phân chia bài toán như vậy được gọi là kỹ thuật quy hoạch động trong lập trình (*Dynamic programming*). Kỹ thuật này rất hiệu quả để giải nhiều bài toán lập trình tối ưu, và lời giải cho bài toán là một minh họa về sử dụng kỹ thuật quy hoạch động.

Ta xét bài toán:

Hãy chọn các số hạng trong i số hạng $a_0, a_1, .., a_i$ sao cho:

- Không chọn hai số hạng kề nhau.
- Tổng các số hạng được chọn là lớn nhất.

Gọi L[i] là tổng lớn nhất của các số hạng được chọn thỏa mãn yêu cầu trên. Khi đó ta có:

$L[0] = a_0$.

$L[1] = \max(a_0, a_1)$.

Bước này được gọi là bước khởi tạo.

L[i] = ? khi i = 2, 3, .., n – 1.

Nhận thấy, trong cách chọn các số hạng thuộc dãy $a_0, a_1, .., a_i$ sẽ có hai trường hợp.

- Không chọn a_i; khi đó các số hạng được chọn thuộc dãy $a_0, a_1, \ldots, a_{i-1}$ và lời giải của bài toán trong trường hợp này là L[i-1].

- Có chọn a_i; như vậy sẽ không chọn a_{i-1}. Các số hạng còn lại nếu được chọn sẽ thuộc dãy $a_0, a_1, .., a_{i-2}$. Các số hạng được chọn trong dãy $a_0, a_1, .., a_{i-2}$ thỏa mãn bài toán là L[i-2]. Như vậy, lời giải của bài toán trong trường hợp này là a_i + max(L[i-2], 0).

Vậy **L[i] = max(L[i-1], a_i + max(0, L[i-2]))**. Công thức này được gọi là công thức truy hồi, (hoặc là công thức quy hoạch động).

Rõ ràng, lời giải của bài toán cần giải là L[n-1].

Chương trình

```
n = int(input("Nhap so so hang n = "))
print("Nhap ",n, " so hang")
a = []
for i in range(n):
    x = int(input())
    a.append(x)
L = []
for i in range(n):
    L.append(0)
L[0] = a[0]
L[1] = max(a[0], a[1])
for i in range(2, n):
    x = max(L[i-2], 0)
    L[i] = max(L[i-1], a[i] + x)
print("Tong gia tri lon nhat: ",L[n-1])
```

CHỦ ĐỀ 11. MỘT SỐ VẤN ĐỀ NÂNG CAO VỀ DANH SÁCH

A. KIẾN THỨC CƠ BẢN

1. Khởi tạo danh sách

Việc khởi tạo một danh sách ta có thể liệt kê tuần tự các phần tử của danh sách. Khi danh sách có nhiều phần tử được tạo ra theo một quy tắc nào đó, ta có thể sử dụng vòng lặp để tạo từng phần tử và thêm chúng vào danh sách.

Ví dụ: Tạo danh sách rỗng [], có hai phần tử ['Lap trinh', 'Python'],.. Tạo một danh sách gồm 100 phần tử 0, 1, 2,.., 99, ta có thể thực hiện:

```
list = []
for i in range(100): list.append(i)
```

Ngoài những cách trên, Python còn cung cấp nhiều cách linh hoạt để tạo ra danh sách. Sau đây là một số ví dụ về tạo ra các danh sách.

- Sử dụng vòng ***for*** để tạo danh sách:

Ví dụ 1. Chương trình:

```
list1 = [i   for i in range(6)]
list2 = [i*i for i in range(6)]
list3 = [0   for i in range(6)]
list4 = [2*i for i in (1, 2, 3)]
print(list1)
print(list2)
print(list3)
print(list3)
```

☞ Kết quả:

```
[0, 1, 2, 3, 4, 5]
[0, 1, 4, 9, 16, 25]
[0, 0, 0, 0, 0, 0]
[2, 4, 6]
```

Cú pháp 1

[<*biểu thức*> **for** <*phần tử*> **in** <*tập giá trị*>]

Tạo một danh sách gồm các phần tử là giá trị của <*biểu thức*> khi <*phần tử*> lần lượt nhận giá trị của <*tập giá trị*>.

Ngoài ra, ta có thêm các điều kiện của <*biểu thức*> để thêm vào danh sách.

Cú pháp 2

[<*biểu thức*> **for** <*phần tử*> **in** <*tập giá trị*> **if** <*điều kiện*>]

Ví dụ 2. chương trình:

```
list1 = [i   for i in range(6) if i%2==0]
list2 = [i*i for i in range(6) if i%2==0]
list3 = [0   for i in range(6) if i%2==0]
list4 = [2*i for i in (1, 2, 3)if i%2==0]
print(list1)
print(list2)
print(list3)
print(list4)
```

☞ Kết quả:

```
[0, 2, 4]
[0, 4, 16]
[0, 0, 0]
[4]
```

- Sử dụng nhiều vòng **for** để tạo danh sách:

Mỗi phần tử của danh sách có thể là các danh sách, khi đó để tạo danh sách, ta có thể sử dụng các vòng for lồng nhau.

Ví dụ 3. Chương trình:

```
list1 = [[i,j] for i in (1, 2, 3) for j in
range(2)]
```

```
list2 = [[i,j] for i in (1, 2, 3)
                    for j in range(2)
                      if i+j== 2
         ]
print(list1)
print(list2)
```

☞ Kết quả:

```
[[1, 0], [1, 1], [2, 0], [2, 1], [3, 0], [3, 1]]
[[1, 1], [2, 0]]
```

- Sử dụng hàm tạo **list()**

Ví dụ 4. Chương trình:

```
st = "Python"
list1 = list(st)
print(list1)
```

☞ Kết quả:

```
['P', 'y', 't', 'h', 'o', 'n']
```

2. Đếm số lần xuất hiện của một phần tử có trong danh sách

Phương thức <*danh sách*>.**count(x)**

Đếm số lần xuất hiện của một phần tử x có trong <*danh sách*>.

Ví dụ 5. Chương trình:

```
list1 = [1, 2, 3, 1, "toan", "ly", "hoa"]
x = 1
y = "toan"
print("so lan xuat hien cua x: ",list1.count(x))
print("so lan xuat hien cua y: ",list1.count(y))
```

☞ Kết quả:

```
so lan xuat hien cua x:  2
so lan xuat hien cua y:  1
```

Ví dụ 6. Chương trình:

```
x = [1, 2, 1, 3].count(1)
print(x)
```

☞ Kết quả:

```
2
```

3. Xóa phần tử khỏi danh sách

Python cung cấp nhiều cách để thực hiện xóa một phần tử khỏi danh sách.

- Phương thức **remove()**

  <*danh sách*>.**remove**(*x*)

Xóa phần tử *x* xuất hiện đầu tiên trong <*danh sách*>. Nếu trong <*danh sách*> không có phần tử *x* thì trình thông dịch sẽ báo lỗi.

Ví dụ 7. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
list1.remove(1)  #xoa phan tu dau tien bang 1
print(list1)
```

☞ Kết quả:

```
[2, 3, 4, 1]
```

- Phương thức **pop()**

  <*danh sách*>.**pop()**

Xóa phần tử cuối cùng của <*danh sách*>.

Ví dụ 8. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
list1.pop()      #lay phan tu cuoi trong danh sach
print(list1)
```

☞ Kết quả:

```
[1, 2, 3, 4]
```

- Hàm **del**

  Hàm **del** <*danh sách*>[*index*]

Xóa phần tử có chỉ số *index* trong <*danh sách*>.

Ví dụ 9. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
del list1[1]          #xoa phan tu co chi so 1
print(list1)
```

☞ Kết quả:

```
[1, 3, 4, 1]
```

- Hàm **del** <*danh sách*>[*i*: *j*]

Xóa các phần tử có chỉ số $i, i+1, .., j-1$ trong <*danh sách*>.

Ví dụ 10. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
del list1[0:3]
print(list1)
```

☞ Kết quả:

```
[4, 1]
```

- Hàm **del** <*danh sách*>[*i*: *j*: *t*]

Xóa các phần tử có chỉ số $i, i+t,\ i+2t, .., i+lt$ trong <*danh sác*
với $i+lt < j$.

Ví dụ 11. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
del list1[0:4:2]
print(list1)
```

☞ Kết quả:

```
[2, 4, 1]
```

- Phương thức **clear()**

  <*danh sách*>.**clear()**

Xóa tất cả các phần tử trong <*danh sách*>. Khi đó danh sách trở thành danh sách rỗng (số phần tử bằng 0).

5. Chèn phần tử vào danh sách

- Phương thức **insert()**

  <*danh sách*>.**insert**(*i*, x)

Chèn phần tử x vào vị trí có chỉ số *i* trong danh sách <*danh sách*>.

Ví dụ 12. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
list1.insert(1,6)
print(list1)
```

☞ Kết quả:

```
[1, 6, 2, 3, 4, 1]
```

6. Sắp xếp các phần tử trong danh sách

- Sắp xếp tăng dần:

  <*danh sách*>.**sort()**

Sắp xếp các phần tử trong <*danh sách*> theo thứ tự tăng dần.

- Sắp xếp giảm dần:

  <*danh sách*>.**sort**(*reverse = True*)

Sắp xếp các phần tử trong <*danh sách*> theo thứ tự giảm dần.

Ví dụ 13. Chương trình:

```
list1 = [1, 2, 3, 4, 1]
list2 = [1, 2, 3, 4, 1]
list1.sort()
list2.sort(reverse = True)
print(list1)
print(list2)
```

☛ Kết quả:

```
[1, 1, 2, 3, 4]
[4, 3, 2, 1, 1]
```

- Hàm **sorted()**

Hàm **sorted(**<*danh sách*>**)** trả về một danh sách mới gồm c phần tử của <*danh sách*> đã được sắp xếp tăng dần.

Hàm **sorted(**<*danh sách*>**,** *reverse* = *True*) trả về một danh sá mới gồm các phần tử của <*danh sách*> đã được sắp xếp giảm dần.

Ví dụ 14. Chương trình:

```
list1 = [5, 2, 3, 4]
list2 = sorted(list1)
list3 = sorted(list1,reverse = True)
print(list1)
print(list2)
print(list3)
```

☛ Kết quả:

```
[5, 2, 3, 4]
[2, 3, 4, 5]
[5, 4, 3, 2]
```

7. Đảo ngược danh sách

Phương thức <*danh sách*>.**reverse()**:

Đảo ngược thứ tự các phần tử trong <*danh sách*>.

Ví dụ 15. Chương trình:

```
list1 = [1, 2, 3, 4, 5]
list1.reverse()
print(list1)
```

☛ Kết quả:

```
[5, 4, 3, 2, 1]
```

❖ ***Chú ý:***

Ta có thể tạo ra một danh sách đảo ngược bằng lệnh *<danh sách>*[::−1].

Ví dụ 16. Chương trình:

```
list1 = [1, 2, 3, 4, 5]
list2 = list1[::-1]
list1.reverse()
print(list1)
print(list2)
```

☞ Kết quả:

```
[5, 4, 3, 2, 1]
[5, 4, 3, 2, 1]
```

8. Ma trận – danh sách hai chiều

Trong Python, các phần tử trong danh sách có thể là một danh sách. Một ma trận hay một mảng hai chiều được biểu diễn như một danh sách mà mỗi phần tử của nó là một danh sách biểu diễn các phần tử trên một dòng.

Với một ma trận gồm 3 dòng, 3 cột như dưới đây được biểu diễn bằng danh sách: [[1, 2, 3], [2, 3, 4], [3, 4, 5]].

1	2	3
2	3	4
3	4	5

Các dòng và các cột trong ma trận được đánh chỉ số bắt đầu từ 0. Để truy cập đến một phần tử nào đó trong bảng ta thực hiện:

<danh sách>[*chỉ số dòng*][*chỉ số cột*].

🖮 Cách in các phần tử của một ma trận.

Ví dụ 17. Chương trình:

```
list1 = [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
for i in range(3):
```

```
        for j in range(3):
            print(list1[i][j], end = " ")
        print()
```

☞ Kết quả:

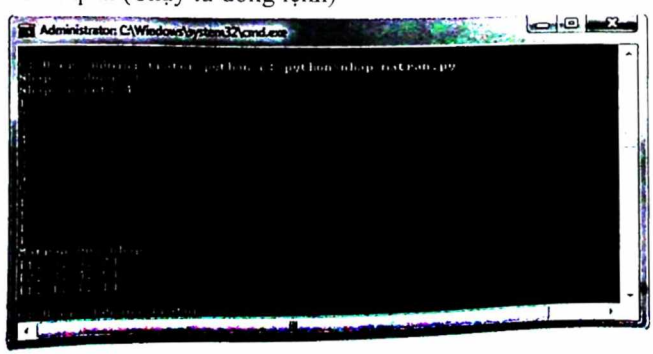
```
1 2 3
2 3 4
3 4 5
```

Ví dụ 18. Viết chương trình nhập vào một ma trận gồm m dòng và n cột. In ra ma trận được nhập vào.

Chương trình:

```
m = int(input("Nhap so dong: "))
n = int(input("Nhap so cot: "))
A = []                          #tao matran rong
for i in range(m):
    row = []                #tao dong rong
    for j in range(n):
        x = int(input())
        row.append(x)
    A.append(row)           #them dong vao matran
print("Matran vua nhap:")
for x in A:
    print(x)
```

☞ Kết quả: (Chạy từ dòng lệnh)



❖ *Chú ý*

Ta có thể nhập một danh sách (một chiều hoặc nhiều chiều) bằng sử dụng vòng for như sau:

```
m = int(input("Nhap so dong: "))
n = int(input("Nhap so cot: "))
#tao matran A bang vong lap for
A = [[int(input()) for j in range(n)] for i in range(
print("Matran vua nhap: ")
for x in A:
    print(x)
```

Vòng lặp `[int(input()) for j in range(n)]` *dùng để nhập n phần tử trong một hàng của ma trận.*

`[[int(input()) for j in range(n)] for i in range(m)]` *sẽ lặp m lần, mỗi lần sẽ thực hiện vòng lặp* `[int(input()) for j in range(n)]`, *tức là nhập một hàng. Như vậy sẽ nhập được m hàng, từ đó ta sẽ có một ma trận.*

9. Danh sách của các danh sách

Ma trận hai chiều như ở trên là trường hợp riêng của danh sách mà các phần tử của nó là các danh sách. Mỗi phần tử trong danh sách là các danh sách mà số phần tử của nó có thể không bằng nhau.

Ví dụ 19. Chương trình:

```
li1 = [1, 2, 3]
li2 = ['a', 'b', 'c']
li3 = ['python']
li4 = [2020]
list1 = [li1, li2, li3]
list1.append(li4)
print("list1: ",list1)
```

```
print("cac phan tu trong list1:")
for x in list1:
    for y in x:
        print(y, end = " ")
    print()
```

☛ Kết quả:

```
list1:  [[1, 2, 3], ['a', 'b', 'c'], ['python'], [2020]]
cac phan tu trong list1:
1 2 3
a b c
python
2020
```

❖ ***Chú ý***

Đoạn lệnh in ra các phần tử trong danh sách ở chương trình trên:

```
print("cac phan tu trong list1:")
for x in list1:
    for y in x:
        print(y, end = " ")
    print()
```

Cách viết khác:

```
print("cac phan tu trong list1:")
n = len(list1)
for i in range(n):
    m = len(list1[i])
    for j in range (m):
        print(list1[i][j], end = " ")
    print()
```

10. So sánh hai danh sách

Python cho phép so sánh hai danh sách bằng cách lần lượt thực hiện so sánh các phần tử từ chỉ số 0 đến hết danh sách. Khi gặp hai phần tử tương ứng khác nhau thì danh sách nào chứa phần tử lớn hơn thì danh sách đó lớn hơn.

Ví dụ 20. Chương trình:

```
list1 = [1, 2, 3]
list2 = [1, 3, 2]
list3 = [1, 2, 3, 4]
print(list1 == list2)
print(list1 < list2)
print(list1 < list3)
```

☞ Kết quả:

```
False
True
True
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Xoay vòng danh sách

Cho một danh sách list. Hãy xoay vòng danh sách bằng cách chuyển phần tử cuối cùng đến vị trí đầu tiên của danh sách. Ví dụ: [1, 2, 3,4, 5] *xoay vòng thành* [5, 1, 2, 3, 4].

Bài tập 2. Dãy các phần tử liên tiếp bằng 0 và dài nhất

Tạo một danh sách có n phần tử bằng cách sinh ngẫu nhiên (dùng hàm randint(0, 1)), *các phần tử là số* 0 *hoặc* 1. *Tìm dãy gồm các phần tử liên tiếp có giá trị đều bằng* 0 *và có nhiều phần tử nhất. Ví dụ* [1, 0, 0, 0, 1, 0], *dãy gồm các phần tử liên tiếp có giá trị bằng* 0 *và nhiều phần tử nhất là:* 0, 0, 0; *có* 3 *phần tử.*

Bài tập 3. Tổng các số hạng liên tiếp trong dãy có giá trị lớn nhất

Cho dãy các số nguyên, hãy tìm các số hạng liên tiếp của dãy sao cho tổng các số hạng đó đạt giá trị lớn nhất.

Ví dụ: Dãy 1, -10, 2, 3, -2, 10. *Các số hạng liên tiếp được chọn để tổng có giá trị lớn nhất là:* 2, 3, -2, 10, *tổng lớn nhất bằng* 2 + 3 + (-2) + 10 = 13.

Bài tập 4. Bài toán thống kê

Tạo một danh sách A có n số nguyên ($n \leq 10^6$) bằng cách sinh ngẫu nhiên. Các số có giá trị thuộc [0; 10^6]. Hãy thống kê danh sách xem, mỗi phần tử xuất hiện bao nhiêu lần.

Ví dụ: A = [1, 2, 3, 1, 3, 6], *thống kê:*

Phần tử 1 xuất hiện 2 lần;

Phần tử 2 xuất hiện 1 lần;

Phần tử 3 xuất hiện 2 lần;

Phần tử 6 xuất hiện 1 lần.

Bài tập 5. Gộp hai danh sách

Cho hai danh sách list1 và list2 có các phần tử được sắp xếp tăng dần. Hãy gộp hai danh sách này thành danh sách list3 có các phần tử cũng được sắp xếp theo thứ tự tăng dần. Ví dụ: list1 = [1, 1, 2, 9], *list2* = [1, 4, 4, 5], *danh sách list3* = [1, 1, 1, 2, 4, 4, 5, 9].

Bài tập 6. Dãy con tăng dài nhất

Xét dãy số A gồm n số nguyên a_0, a_1, .. , a_{n-1}. Khi xóa một số các số hạng của dãy (có thể không xóa), các số hạng còn lại, vẫn giữa nguyên thứ tự, tạo thành một dãy và được gọi là dãy con của dãy A.

Ví dụ, dãy A gồm: 1, 4, 5, 3, 7. Khi xóa số hạng 4, 3 ta được dãy 1, 5, 7 và được gọi là dãy con của dãy A.

Một dãy gồm các số hạng b_0, b_1, .. , b_k được gọi là dãy tăng nếu $b_0 < b_1 < b_2 < .. < b_k$. Số các số hạng của dãy được gọi là độ dài của dãy đó.

Yêu cầu: Cho dãy A gồm n số nguyên $a_0, a_1, .. , a_{n-1}$. Hãy tìm một dãy con của dãy A thỏa mãn:

- Dãy tăng.

- Có nhiều số hạng nhất (độ dài lớn nhất).

Với dãy A: 1, 4, 5, 3, 7; *dãy con tăng dài nhất tìm được:* 1, 5, 7.

Bài tập 7. Tổng chéo chính và chéo phụ

Nhập vào một bảng số A gồm n dòng và n cột. Tính tổng các số nằm trên đường chéo chính; tính tổng các số nằm trên đường chéo phụ của bảng.

Ví dụ n = 5, bảng được nhập vào:

1	0	0	0	**4**
0	**1**	0	**4**	0
0	0	**1**	0	0
0	**4**	0	**1**	0
4	0	0	0	**1**

Các số nằm trên đường chéo chính: 1, 1, 1, 1, 1; *nên tổng bằng* 5.

Các số nằm trên đường chéo phụ: 4, 4, 1, 4, 4; *nên tổng bằng* 17.

Bài tập 8. Ma trận chuyển vị

Xét ma trận A gồm m dòng và n cột: $A = \begin{bmatrix} a_{11} & a_{12} .. a_{1n} \\ \cdots & \cdots \ .. \ \cdots \\ a_{m1} & a_{m2} .. a_{mn} \end{bmatrix}$. *Ma trận chuyển vị của A được kí hiệu là* A^T *gồm n dòng và m cột, trong đó* $A^T_{ij} = A_{ji}$.

Ví dụ: $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 9 & 8 \end{bmatrix}$, *ma trận chuyển vị* $A^T = \begin{bmatrix} 1 & 2 \\ 2 & 9 \\ 3 & 8 \end{bmatrix}$.

Yêu cầu, viết chương trình nhập vào ma trận A, đưa ra ma trận A^T.

Bài tập 9. Nhân ma trận

Xét ma trận A gồm m dòng và n cột và ma trận B gồm n dòng và k cột.

$$A = \begin{bmatrix} a_{11} & a_{12} .. a_{1n} \\ \cdots & \cdots .. \cdots \\ a_{m1} & a_{m2} .. a_{mn} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} .. a_{1k} \\ \cdots & \cdots .. \cdots \\ b_{n1} & a_{n2} .. a_{nk} \end{bmatrix}$$

Tích của ma trận A và ma trận B là một ma trận C gồm m dòng và k cột.

$$C = \begin{bmatrix} c_{11} & c_{12} .. c_{1k} \\ \cdots & \cdots .. \cdots \\ c_{m1} & c_{m2} .. c_{mk} \end{bmatrix} \text{ trong đó } c_{ij} = \sum_{t=1}^{n} a_{it} \times b_{tj}.$$

Yêu cầu: Viết chương trình nhập ma trận A, B và tính ma trận C.

$$\text{Ví dụ: } A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \rightarrow C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Xoay vòng danh sách

Ý tưởng thuật toán

Lấy phần tử đầu tiên của danh sách *list* là $x = list[0]$.

Xóa phần tử đầu tiên: **del** $list[0]$ hoặc $list.remove(x)$

Thêm x vào cuối danh sách: $list.append(x)$ hoặc $list + [x]$.

Chương trình

```
list = [1, 2, 3, 4, 5]
x = list[0]
del list[0]
list.append(x)
print(list)
```

Bài tập 2. Dãy các phần tử liên tiếp bằng 0 và dài nhất

Ý tưởng thuật toán

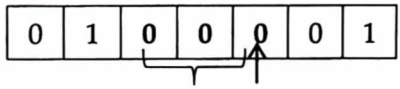
Để tạo ngẫu nhiên một danh sách gồm 100 phần tử bằng 0 hoặc 1. Ta sử dụng hàm **randint(0, 1)** trong môđun **random**.

Sử dụng hai biến kq và tam với ý nghĩa:

Biến kq lưu số phần tử của dãy các phần tử bằng 0 liên tiếp và dài nhất. Ban đầu $kq = 0$.

Biến tam là số phần tử kề liên tiếp với phần tử hiện tại (đang xét) và bằng 0.

Ví dụ:



Phần tử hiện tại, có 3 phần tử kề liên tiếp bằng 0, $tam = 3$.

Lần lượt duyệt qua các phần tử của danh sách.

Khi gặp phần tử bằng 1, ta cho $tam = 0$.

Khi gặp phần tử bằng 0, ta tăng *tam* lên 1 ($tam = tam + 1$) sau đó cập nhật độ dài của dãy các phần tử bằng 0 và dài nhất, tức là $kq = max(kq, tam)$.

Chương trình

```
from random import randint
n = int(input("Nhap so phan tu cua danh sach: n = "))
a = []
for i in range(n):
    x = randint(0,1)
    a.append(x)
print("Danh sach duoc tao ngau nhien:")
print(a)
kq = tam = 0
for x in a:
    if x == 0:
        tam = tam + 1
        kq = max(kq, tam)
    else:
        tam = 0
print("So phan tu cua day dai nhat tim duoc la: ",kq)
```

Bài tập 3. Tổng các số hạng liên tiếp trong dãy có giá trị lớn nhất

Ý tưởng thuật toán:

Xét dãy $a_1, a_2, ..., a_n$. Dãy gồm các số hạng liên tiếp có lạng: $a_i, a_{i+1}, ..., a_j (1 \le i \le j \le n)$ và tổng bằng $a_i + a_{i+1} + .. + a_j$.

Đặt $S_0 = 0, S_i = a_1 + a_2 + .. + a_i$ với $i = 1, 2, .., n$. Khi đó:

$a_i + a_{i+1} + .. + a_j = a_1 + a_2 + .. + a_{i-1} + a_i + a_{i+1} + .. + a_j - (a_1 + a_2 + .. + a_{i-1}) = S_j - S_{i-1}$.

Để tính S_i, có thể thực hiện liên tiếp từng bước với $i = 1, 2, 3, .., n$.

$S_1 = S_0 + a_1; S_2 = S_1 + a_2; .. ; S_i = S_{i-1} + a_i; .. ; S_n = S_{n-1} + a_n$

Ta sẽ duyệt tất cả các giá trị của $j = 1, 2, 3, .., n$. Với mỗi giá trị của j, thì có: $a_i + a_{i+1} + .. + a_j = S_j - S_{i-1}$ đạt giá trị lớn nhất khi S_{i-1} là giá trị nhỏ nhất trong các giá trị $S_0, S_1, .., S_{j-1}$ và có giá trị $min(S[:j])$. Vậy tổng $a_i + a_{i+1} + .. + a_j$ đạt giá trị lớn nhất bằng $x = S[j] - min(S[:j])$. Với mỗi giá trị x tìm được, ta sẽ cập nhật giá trị lớn nhất $kq = max(kq, x)$. Ban đầu $kq = a[1]$.

Chương trình:

```
n = int(input("Nhap so n = "))
print("Nhap ",n," so hang: ")
a = [int(input()) for i in range(n)]
# de cac so hang nhap vao co chi so 1, 2,.., n
a = [0] + a
S = [0]
for i in range(1, n+1):
    S = S + [S[i-1] + a[i]]
kq = a[1]
for j in range(2, n+1):
    x = S[j] - min(S[:j])
    kq = max(kq, x)
print("Tong cac so hang lien tiep lon nhat: ", kq)
```

Một lời giải đẹp:

```
n = int(input("Nhap so n = "))
print("Nhap ",n," so hang: ")
a = [int(input()) for i in range(n)]
# de cac so hang nhap vao co chi so 1, 2,.., n
a = [0] + a
s = 0
kq = a[1]
for i in range(1, n+1):
    s = s + a[i]
    kq = max(kq, s)
    if s < 0: s = 0
print("Tong cac so hang lien tiep lon nhat: ", kq)
```

Bài tập 4. Bài toán thống kê

Ý tưởng thuật toán

Do các phần tử trong danh sách có giá trị thuộc $[0; 10^6]$, nên có thể tạo một danh sách $list$ có $10^6 + 1$ phần tử. Trong đó $list[i]$ là số lần xuất hiện của phần tử có giá trị là i.

Ban đầu $list[i] = 0$ với $i = 1, 2, .., 10^6$.

Lần lượt duyệt tất cả các phần tử của danh sách A. Với một phần tử x, ta sẽ tăng $list[x]$ lên 1, tức là $list[x] = list[x] + 1$.

Cuối cùng, lần lượt xét các giá trị $i = 0, 1, .., 10^6$, nếu $list[i] > 0$ thì phần tử có giá trị i xuất hiện $list[i]$ lần trong dãy A.

Chương trình

```
from random import randint
n   = int(input("Nhap so phan tu cua danh sach "))
A   = [randint(0, 1000000) for i in range(n)]
list = [0 for i in range(1000001)]
for x in A:
```

```
        list[x] += 1
    print("Ket qua thong ke: ")
    for i in range(1000001):
        if list[i] > 0:
        print("Phan tu ",i," xuat hien ", list[i]," lan")
```

Bài tập 5. Gộp hai danh sách

Ý tưởng thuật toán

Thuật toán 1

```
list3 = list1 + list2
list3.sort()
```

Với cách làm trên, ta cũng nhận được danh sách *list*3 gồm các phần tử của danh sách *list*1 và *list*2 và được sắp xếp theo thứ tự tăng dần. Nhưng thuật toán trên chưa sử dụng hết giả thiết là các phần tử trong hai danh sách *list*1 và *list*2 đã được sắp xếp tăng dần. Đồng thời, việc sử dụng phương thức sắp xếp trong danh sách *list*3 làm tăng thời gian thực hiện lên (số phép tính thực hiện khoảng $n \times log_2 n$ với n là số phần tử trong *list*3).

Một thuật toán khác với số phép tính thực hiện khoảng n khi sử dụng giả thiết các phần tử trong *list*1 và *list*2 đã được sắp xếp.

Thuật toán 2

Khởi tạo *list*3 = [].

Duyệt lần lượt đồng thời các phần tử của hai danh sách *list*1 và *list*2 theo thứ tự từ nhỏ đến lớn.

Gọi x là phần tử đầu tiên trong *list*1, y là phần tử đầu tiên trong *list*2.

Nếu x < y thì cho x vào *list*3; x nhận giá trị phần tử kế tiếp trong *list*1.

Ngược lại cho y vào *list*3; y nhận giá trị phần tử kế tiếp trong *list*2.

Cứ thực hiện như vậy cho đến khi một danh sách hết, một danh sách còn. Đưa lần lượt các phần tử trong danh sách còn vào *list3*.

Chương trình

```
list1 = [1, 2, 3, 9, 10, 10]
list2 = [1, 4, 5, 7]
list3 = []
n = len(list1)
m = len(list2)
i = j = 0
while i < n and j < m:
   if list1[i] < list2[j]:
      list3.append(list1[i])
      i +=1
   else:
      list3.append(list2[j])
      j +=1
#mot danh sach het, mot danh sach con
while i < n:
   list3.append(list1[i])
   i += 1
while j < m:
   list3.append(list2[j])
print(list3)
```

Bài tập 6. Dãy con tăng dài nhất

Ý tưởng thuật toán

Ta có thể sử dụng kỹ thuật quy hoạch động để giải bài toán này.

Sử dụng một danh sách L gồm n phần tử với ý nghĩa: $L[i]$ là độ dài (số phần tử) của dãy con tăng dài nhất của dãy A mà phần tử đầu tiên là $a[i]$ với $i = 0, 1, .., n - 1$.

Việc tính giá trị L[i], sẽ được thực hiện tuần tự với i = n – 1, i = n – 2 .. , i = 0.

Nhận thấy L[n-1] = 1 vì dãy chỉ gồm một phần tử a[n-1].

Với mỗi i = n – 2, n – 3, .. , 1, 0, ta có hai trường hợp:

- Dãy chỉ có một phần tử a[i], trường hợp này $L[i] = 1$.

- Dãy gồm a[i] và các phần tử khác nữa a[t], ...

Trong trường hợp này, a[i] < a[t] và i < t. Hơn nữa để dãy a[i], a[t], ... dài nhất thì dãy a[t], .. cũng phải dài nhất và giá trị độ dài đó là L[t]. Do vậy L[i] = 1 + L[t].

Công thức quy hoạch động:

L[i] = max(1 + L[t]) với t = i+1, .., n – 1 và a[i] < a[t].

Độ dài của dãy con tăng dài nhất là max(L[0], L[1], .. , L[n-1]).

Chương trình

```
n = int(input("Nhap so phan tu cua day: "))
print("Nhap ",n," phan tu: ")
a = [int(input()) for i in range(n)]
L = [1 for i in range(n)]
i = n - 2
while  i >= 0:
   for t in range(i+1,n):
      if a[i] < a[t]:
         L[i] = max(L[i], 1 + L[t])
   i -=1
print("Do dai cua day con tang dai nhat:
",max(L))
```

Bài tập 7. Tổng chéo chính và chéo phụ

Ý tưởng thuật toán

Ta nhận thấy:

Các phần tử nằm trên đường chéo chính có chỉ số A[0][0] A[1][1], .. , A[n-1][n-1].

Các phần tử nằm trên đường chéo phụ có chỉ số A[n-1][0], A[n-2][1], .. , A[0][n-1].

Để tính tổng các phần tử trên đường chéo chính, đường chéo phụ, ta chỉ việc cộng các phần tử có chỉ số tương ứng như trên.

Chương trình

```
n = int(input("Nhap n =  "))
print("Nhap bang so")
A = []
for i in range(n):
    row = [int (input()) for j in range(n)]
    A.append(row)
cheo_chinh = 0
cheo_phu = 0
for i in range(n):
    cheo_chinh += A[i][i]
for i in range(n):
    cheo_phu += A[i][n-i-1]
print("Tong cheo chinh ", cheo_chinh)
print("Tong cheo phu ", cheo_phu)
```

Bài tập 8. Ma trận chuyển vị

Ý tưởng thuật toán

- Nhập m, n và ma trận A gồm m dòng và n cột.
- Tạo ma trận AT gồm n dòng và m cột, các phần tử đều bằng 0.
- Thực hiện gán giá trị $AT[i][j] = A[j][i]$ với $0 \leq i < n, 0 \leq j < m$.

Chương trình

```
m = int(input("Nhap m = "))
n = int(input("Nhap n = "))
print("Nhap matran A: ", m, " dong ", n, " cot ")
```

```
A = [[int(input()) for i in range(n)] for j in range(m)
#Tao ma tran AT gom n dong va m cot, cac phan tu = 0
AT = []
for i in range (n):
    row = []
    for j in range(m):
        row.append(0)
    AT.append(row)
#Tao ma tran chuyen vi
for i in range(n):
    for j in range(m):
        AT[i][j] = A[j][i]
print("Ma tran A: ")
for x in A:
    print(x)
for x in AT:
    print(x)
```

Bài tập 9. Nhân ma trận

Ý tưởng thuật toán

Để tính ma trận C, ta thực hiện tính từng phần tử $c_{ij} = \sum_{t=1}^{n} a_{it} \times b_{tj}$.

Chương trình:

```
m = int(input("Nhap m = "))
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
print("Nhap matran A: ", m, " dong ", n, " cot ")
A = [[int(input()) for i in range(n)] for j in range(m
print("Nhap matran B: ", n, " dong ", k, " cot ")
```

```
B = [[int(input()) for i in range(k)] for j in range(n)
#tao matran C gom m dong, k cot, cac phan tu deu bang 0
C = [[0 for i in range(k)] for j in range(m)]
for i in range(m):
    for j in range(k):
        for t in range(n):
            C[i][j] += A[i][t]*B[t][j]
print("Ma tran A: ")
for x in A:
    print(x)
print("Ma tran B: ")
for x in B:
    print(x)
print("Ma tran C: ")
for x in C:
    print(x)
```

CHỦ ĐỀ 12. KIỂU DỮ LIỆU BỘ - TUPLE

A. KIẾN THỨC CƠ BẢN

Kiểu dữ liệu bộ (tuple) có nhiều điểm giống với kiểu dữ liệu
sách. Tuy nhiên, ở một số khía cạnh nào đó kiểu dữ liệu tuple c
điểm riêng so với kiểu dữ liệu list.

1. Khai báo kiểu dữ liệu bộ - tuple

<biến tuple> = (<phần tử 0>, <phần tử 1>, .. , <phần tử n-1>

Nếu bộ chỉ có một phần tử, thì cần khai báo:

<biến tuple> = (<phần tử 0>,)

Ví dụ 1. Chương trình:

```
tuple1 = (1, 2, "Python", 3, 4)
tuple2 = (1)
tuple3 = (1,)
print(tuple1)
print(tuple2)
print(tuple3)
```

☛ Kết quả:

```
(1, 2, 'Python', 3, 4)
1
(1,)
```

Như vậy (1) được xem như là 1.

2. Cách đánh chỉ số các phần tử

Cách đánh chỉ số các phần tử trong kiểu dữ liệu bộ giống kiểu dữ liệu danh sách và kiểu dữ liệu chuỗi.

Các phần tử được đánh chỉ số từ $0, 1, .., n-1$ theo hướng từ sang phải và $-1, -2, .., -n$ theo hướng từ phải sang trái. Ở đây số phần tử.

3. Hàm lấy số phần tử và cách truy cập đến các phần tử

- Hàm lấy số phần tử

`len(`*tuple*1`)`

Trả về số phần tử trong *tuple*1.

- Truy cập đến các phần tử

*tuple*1[*i*] truy cập đến phần tử có chỉ số *i* của *tuple1*.

Ví dụ 2. Chương trình:

```
tuple1 = (0, 1, 2, 3, 4)
n = len(tuple1)
print("So phan tu cua tuple1: ", n)
print("Cac phan tu:")
for i in range(n):
    print(tuple1[i],end = ' ')
print()
```

☞ Kết quả:

```
So phan tu cua tuple1:  5
Cac phan tu:
0 1 2 3 4
```

4. Lấy các phần tử liên tiếp trong bộ

- Phương thức *tuple*1[*a*: *b*]

Trả về một bộ gồm các phần tử có chỉ số $a, a+1, .., b-1$ của bộ *tuple*1.

- Phương thức *tuple*1[: *b*]

Trả về một bộ gồm b phần tử đầu tiên, tức là các phần tử có chỉ số $0, 1, .., b-1$ của bộ *tuple*1.

- Phương thức *tuple*1[b:]

Trả về một bộ gồm các phần tử có chỉ số $b, b+1, ..$ đến hết của bộ.

Ví dụ 3. Chương trình:

```
tuple1 = ('a', 'b', 'c', 'd', 'e', 'f', 'g')
print(tuple1[1:3])
print(tuple1[:3])
print(tuple1[3:])
```

☞ Kết quả:

```
('b', 'c')
('a', 'b', 'c')
('d', 'e', 'f', 'g')
```

5. Nối (ghép) và lặp các bộ

Nối các bộ

<tuple 1> + <tuple 2> + . . + <tuple n>

Trả về một bộ gồm các phần tử (theo thứ tự) của các bộ *<tuple 1>*, *<tuple 2>*, . . , *<tuple n>*

Ví dụ 4. Chương trình:

```
tuple1 = ('a', 'b', 'c')
tuple2 = (1, 2, 3)
tuple3 = ('a', 'b')
tuple4 =  tuple1 + tuple2 + tuple3
print(tuple4)
```

☞ Kết quả:

```
('a', 'b', 'c', 1, 2, 3, 'a', 'b')
```

Lặp các bộ

*<tuple 1> *n*

n<tuple 1>*

Trả về một bộ bằng cách ghép liên tiếp *n* bộ *<tuple 1>* lại với nhau.

Ví dụ 5. Chương trình:

```
tuple1 = (1, 2, 3)
tuple2 = 2*tuple1
tuple3 = tuple1*2
print(tuple2)
print(tuple3)
```

☞ Kết quả:

```
(1, 2, 3, 1, 2, 3)
(1, 2, 3, 1, 2, 3)
```

6. Một số cách gán giá trị cho kiểu dữ liệu bộ

Sử dụng hàm tạo – `tuple()`

Ta có thể sử dụng hàm tạo **`tuple()`** để tạo một bộ từ một danh sách hoặc từ một chuỗi.

Ví dụ 6. Chương trình:

```
list1 = [1, 2, 3, 4]
str1  = "abcd"
tuple1 = tuple(list1)
tuple2 = tuple(str1)
tuple3 = tuple(range(4))
print(tuple1)
print(tuple2)
print(tuple3)
```

☞ Kết quả:

```
(1, 2, 3, 4)
('a', 'b', 'c', 'd')
(0, 1, 2, 3)
```

Packing và Unpacking

Ví dụ 7. Chương trình:

```
tuple1 = 1, 2, 3        #packing
print(tuple1)
x1, x2, x3 = tuple1     #unpacking
print(x1)
print(x2)
print(x3)
```

☞ Kết quả:

```
(1, 2, 3)
1
2
3
```

Ví dụ 8. Chương trình:

```
a = 1
print(a)
a = 1,
print(a)
```

☞ Kết quả:

```
1
(1,)
```

7. Đảo ngược thứ tự

Ví dụ 9. Chương trình:

```
tuple1 = (1, 2, 3, 4)
tuple2 = tuple1[::-1]
print("Tuple ban dau: ",tuple1)
print("Tuple dao nguoc: ",tuple2)
```

☞ Kết quả:

```
Tuple ban dau:  (1, 2, 3, 4)
Tuble dao nguoc:  (4, 3, 2, 1)
```

8. Hàm min(), max()

- Hàm **min**(*tuple1*)

Trả về phần tử có giá trị nhỏ nhất trong *tuple1*.

- Hàm **max**(*tuple1*)

Trả về phần tử có giá trị lớn nhất trong *tuple1*.

Ví dụ 10. Chương trình:

```
tuple1 = (1, 2, 3, 4)
print("Phan tu nho nhat: ", min(tuple1))
print("Phan tu lon nhat: ", max(tuple1))
```

☞ Kết quả:

```
Phan tu nho nhat:  1
Phan tu lon nhat:  4
```

9. Phép toán in, phương thức count(), phương thức index()

- Phép toán **in**

<phần tử> **in** *<tuple1>*

Trả về giá trị True nếu *<phần tử>* thuộc *<tuple1>*. Ngược lại phép toán trả về False.

- Phương thức **count()**

tuple1.**count**(*<phần tử>*)

Trả về số lần xuất hiện của *<phần tử>* trong *tuple1*.

- Phương thức **index()**

tuple1.**index**(*<phần tử>*)

Trả về chỉ số của *<phần tử>* đầu tiên trong *tuple1*. Nếu *<phần tử>* không thuộc *tuple1* thì chương trình sẽ báo lỗi.

Ví dụ 11. Chương trình:

```
tuple1 = (2, 1, 2, 2, 1)
print(1 in tuple1)
print(5 in tuple1)
print(tuple1.count(2))
print(tuple1.index(2))
```

☛ Kết quả:

```
True
False
3
0
```

10. So sánh bộ

Ví dụ 12. Chương trình:

```
tuple1 = (1, 2, 3)
tuple2 = (2, 1, 3)
tuple3 = (1, 2, 3, 4)
print(tuple1 == tuple2)
print(tuple1 < tuple2)
print(tuple1 < tuple3)
```

☛ Kết quả:

```
False
True
True
```

11. Một số vấn đề không thực hiện được trong kiểu dữ liệu bộ

- Thay đổi giá trị của một thành phần

Kiểu dữ liệu bộ không cho phép thay đổi giá trị của một thành phần.

```
tuple1 = (1, 2, 3, 4)
tuple1[0] = 10
print(tuple1)
```

Chương trình sẽ báo lỗi tại câu lệnh: `tuple1[0] = 10`

- Thêm một phần tử mới

Kiểu dữ liệu bộ không có phương thức thêm một phần tử mới. Nhưng ta có thể sử dụng cách nối bộ để thêm một phần tử mới.

```
tuple1 = (1, 2, 3, 4)
tuple1 = tuple1 + (x,)
```

- Xóa một phần tử

Kiểu dữ liệu bộ cũng không cho phép xóa các phần tử.

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Nhập vào một bộ

Viết chương trình nhập vào một bộ (tuple) gồm số n số nguyên (a_0, a_1, .., a_{n-1}). Tính tổng các phần tử là số nguyên lẻ; tính tổng các phần tử chẵn của bộ.

Bài tập 2. Sắp xếp các phần tử có hai thông tin

Có n đồ vật, đồ vật thứ i ($i = 0, 1, 2, .., n - 1$) có khối lượng là a_i và giá trị là b_i. Hãy sắp xếp các đồ vật theo tiêu chí:

- Khối lượng nhỏ thì xếp trước, khối lượng lớn thì xếp sau (ưu tiên a_i).

- Hai vật có khối lượng bằng nhau thì vật nào có giá trị nhỏ thì xếp trước (ưu tiên b_i).

Chương trình nhập n số nguyên a_0, a_1, .. , a_{n-1} và n số b_0, b_1, .. , b_{n-1} tương ứng là khối lượng của n đồ vật và giá trị của n đồ vật. In ra danh sách n đồ vật sau khi sắp xếp. Danh sách in trên n dòng, mỗi dòng gồm hai số tương ứng là khối lượng và giá trị của một đồ vật.

Ví dụ: n = 4; dãy a_i: 1, 2, 9, 2; dãy b_i: 10, 3, 2, 1. Danh sách sau khi sắp xếp.

1 10

2 1

2 3

9 2

Bài tập 3. Sắp xếp theo trường thông tin

Có n học sinh, học sinh thứ i ($i = 0,1,..,n-1$) có điểm Toán là a_i, điểm Tin là b_i và điểm Anh là c_i. Hãy sắp xếp danh sách học sinh theo điểm Tin tăng dần.

Ví dụ: $n = 4$; dãy a_i: 10, 6, 7, 9; dãy b_i: 10, 9, 7, 8; dãy c_i: 6, 8, 7, 9. Danh sách sau khi sắp xếp (Toán, Tin, Anh).

7 7 7

9 8 9

6 9 8

10 10 6

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Nhập vào một bộ

Ý tưởng thuật toán

Để nhập một bộ, ta có thể thực hiện theo hai cách:

- ***Cách 1***. Nhập các thành phần và lưu các thành phần này trong một danh sách. Sau đó sử dụng hàm tạo ***tuple***() để chuyển danh sách các phần tử được nhập thành một bộ.
- ***Cách 2.*** Tạo một bộ *tuple*1 = (). Lần lượt nhập các thành phần, lưu chúng thành bộ gồm 1 phần tử rồi ghép chúng vào *tuple*1.

Để tính tổng các phần tử lẻ, tổng các phần tử chẵn, ta xét lần lượt các phần tử, kiểm tra tính chẵn lẻ rồi cộng vào tổng tương ứng.

Chương trình 1

```
n = int(input("Nhap so phan tu cua tuple: "))
print("Nhap ", n, "phan tu: ")
list1 = [int(input()) for i in range(n)]
tuple1 = tuple(list1)
print("tuple nhap vao: ",tuple1)
sum_odd = sum_even = 0
```

```
for x in tuple1:
    if x%2 == 0: sum_even += x
    else: sum_odd += x
print("Tong cac phan tu le: ",sum_odd)
print("Tong cac phan tu chan: ",sum_even)
```

Chương trình 2

```
n = int(input("Nhap so phan tu cua tuple: "))
print("Nhap ", n, "phan tu: ")
tuple1 = ()
for i in range(n):
    x = int(input())
    tuple1 += (x,)
print("tuple nhap vao: ",tuple1)
sum_odd = sum_even = 0
for x in tuple1:
    if x%2 == 0: sum_even += x
    else: sum_odd += x
print("Tong cac phan tu le: ",sum_odd)
print("Tong cac phan tu chan: ",sum_even)
```

Bài tập 2. Sắp xếp các phần tử có hai thông tin

Ý tưởng thuật toán

Lưu thông tin của n đồ vật trong một danh sách *list*1 mà mỗi phần tử của danh sách là một bộ gồm (a_i, b_i). Thực hiện sắp xếp danh sách theo thứ tự tăng dần và in ra danh sách đã được sắp xếp.

Lưu dãy $a_0, a_1, .., a_{n-1}$ dưới dạng một danh sách *listA*.

Lưu dãy $b_0, b_1, .., b_{n-1}$ dưới dạng một danh sách *listB*.

Để tạo danh sách *list*1 ta có thể thực hiện:

Cách 1

- Đặt $list1 = []$

- Lần lượt ghép: $list1 = list1 + [(listA[i], listB[i])]$ với $i = 0, 1, .., n - 1$.

Cách 2

Sử dụng hàm **zip()**

Chương trình 1

```
n = int(input("Nhap so do vat n = "))
list1 = []
print("Nhap ",n, "khoi luong: ")
listA = [int(input()) for i in range(n)]
print("Nhap ",n, "gia tri: ")
listB = [int(input()) for i in range(n)]
for i in range(n):
   x = (listA[i], listB[i])
   list1.append(x)
list1.sort()
print("Danh sach do vat da duoc sap xep:")
for i in range(n):
   print(list1[i][0],' ',list1[i][1])
```

Chương trình 2

```
n = int(input("Nhap so do vat n = "))
list1 = []
print("Nhap ",n, "khoi luong: ")
listA = [int(input()) for i in range(n)]
print("Nhap ",n, "gia tri: ")
listB = [int(input()) for i in range(n)]
list1 = list(zip(listA, listB))
list1.sort()
```

```
print("Danh sach do vat da duoc sap xep:")
for i in range(n):
    print(list1[i][0],' ',list1[i][1])
```

Bài tập 3. Sắp xếp theo trường thông tin

Ý tưởng thuật toán:

Lưu thông tin của n học sinh trong một danh sách *list*1 mà mỗi phần tử của danh sách là một bộ gồm (a_i, b_i, c_i).

Lưu dãy $a_0, a_1, .., a_{n-1}$ dưới dạng một danh sách *listA*.

Lưu dãy $b_0, b_1, .., b_{n-1}$ dưới dạng một danh sách *listB*.

Lưu dãy $c_0, c_1, .., c_{n-1}$ dưới dạng một danh sách *listC*.

Khi đó danh sách *list*1 = **list**(**zip**(*listA, listB, listC*))

Để sắp xếp theo trường môn Tin, ta sử dụng khóa **key** và hàm **lambda** như sau: *list*1.sort(**key** = **lambda** hs: hs[1]).

Chú ý: tên hs trong hàm lambda có thể thay bởi một tên bất kì vì đó là một biến, còn hs[1] là vì trường ta cần sắp xếp có chỉ số 1.

Chương trình:

```
n = int(input("Nhap so hoc sinh n = "))
print("Nhap ",n, "diem Toan: ")
listA = [int(input()) for i in range(n)]
print("Nhap ",n, "diem Tin: ")
listB = [int(input()) for i in range(n)]
print("Nhap ",n, "diem Anh: ")
listC = [int(input()) for i in range(n)]
list1 = list(zip(listA, listB, listC))
list1.sort(key = lambda hs: hs[1])
print("Danh sach hoc sinh da duoc sap xep:")
for i in range(n):
print(list1[i][0],list1[i][1], list1[i][2])
```

CHỦ ĐỀ 13. KIỂU DỮ LIỆU TẬP HỢP - SET

A. KIẾN THỨC CƠ BẢN

Ngôn ngữ lập trình Python hỗ trợ kiểu dữ liệu tập hợp (set) cho phép biểu diễn và thực hiện các phép toán trên tập hợp giống như toán học. Đây là một điểm mạnh của ngôn ngữ lập trình Python.

1. Khai báo tập hợp

<biến tập hợp> = {phần tử 0, phần tử 1, .. , phần tử n-1}

Ví dụ: *SetA* = {1, 2, 6, 'toan', 'tin'}

Không giống như trong kiểu dữ liệu danh sách (*list*), các phần tử trong tập hợp không có thứ tự và không được đánh chỉ số. Các phần tử trong tập hợp đều phải khác nhau, giống như khái niệm tập hợp trong toán học.

Ví dụ 1. Chương trình:

```
set1 = {1, 2, 3, 4, 1}
print(set1)
```

☛ Kết quả:

```
{1, 2, 3, 4}
```

Các phần tử bằng nhau được tự động loại bỏ và chỉ giữ lại một.

2. Cách tạo một tập hợp

Để tạo một tập hợp, có thể liệt kê trực tiếp các phần tử của tập hợp. Ngoài ra, Python cho phép tạo ra các tập hợp theo các cách sau.

a) Tạo tập hợp dùng hàm tạo `set()`

Ví dụ 2. Chương trình:

```
list1 = [1, 2, 3, 1, 5]
str1  = "abcacbd"
set1 = set(list1)
```

```
set2 = set(str1)
print(set1)
print(set2)
```

☞ Kết quả:

```
{1, 2, 3, 5}
{'d', 'b', 'a', 'c'}
```

b) Sử dụng vòng for

Có thể sử dụng vòng for để tạo ra tập hợp giống như tạo ra các danh sách.

Ví dụ 3. Chương trình:

```
set1 = {x for x in range(10)}
set2 = {2*x for x in range(10)}
set3 = {x for x in range(10) if x%2 == 0}
print(set1)
print(set2)
print(set3)
```

☞ Kết quả:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
{0, 2, 4, 6, 8}
```

❖ ***Chú ý***

set1 = { } *là một câu lệnh sai,* ***{ }*** *không dùng để biểu diễn một tập rỗng. Để biểu diễn một tập hợp rỗng, ta dùng* ***set1 = set()****.*

3. Lấy số phần tử và liệt kê các phần tử thuộc tập hợp

Ví dụ 4. Chương trình:

```
set1 = {1, 2, 3, 4, 5}
n = Len(set1)
```

```
print("so phan tu cua tap hop: ", n)
print("cac phan tu thuoc tap hop:", end = ' ')
for x in set1:
    print(x,end = ' ')
```

☞ Kết quả:

```
so phan tu cua tap hop:  5
cac phan tu thuoc tap hop: 1 2 3 4 5
```

- Hàm lấy số phần tử của tập hợp

len(<*biến tập hợp*>)

4. Các phép toán trên tập hợp

a) Phép giao

Giao hai tập hợp *A* và *B* là một tập hợp, bao gồm tất cả các phần tử cùng thuộc cả hai tập hợp *A* và *B*.

Kí hiệu: ***A*** **&** ***B***

Ví dụ 5. Chương trình:

```
A = {1, 2, 3, 4}
B = {2, 4, 6, 7}
C = A & B
print("Giao cua A va B: ",C)
```

☞ Kết quả:

```
Giao cua A va B:  {2, 4}
```

- Phương thức **intersection()**

Để tìm giao của hai tập hợp, ta sử dụng phương thức **intersection()**.

Ví dụ 6. Chương trình:

```
A = {1, 2, 3, 4}
B = {2, 4, 6, 7}
```

```
C = A.intersection(B)
D = B.intersection(A)
print("Giao cua A va B: ", C)
print("Giao cua A va B: ", D)
```

☞ Kết quả:

```
Giao cua A va B:  {2, 4}
Giao cua A va B:  {2, 4}
```

b) Phép hợp

Hợp của hai tập hợp *A* và *B* là một tập hợp bao gồm tất cả các phần tử thuộc tập *A* hoặc tập *B*.

Kí hiệu: **A | B**

Ví dụ 7. Chương trình:

```
A = {1, 2, 3, 4}
B = {2, 4, 6, 7}
C = A | B
print("Hop cua A va B: ", C)
```

☞ Kết quả:

```
Hop cua A va B:  {1, 2, 3, 4, 6, 7}
```

➧ Phương thức **union()**

Để tìm hợp của hai tập hợp, ta có thể sử dụng phương thức **union()**.

Ví dụ 8. Chương trình:

```
A = {1, 2, 3, 4}
B = {2, 4, 6, 7}
C = A.union(B)
D = B.union(A)
print("Hop cua A va B: ", C)
print("Hop cua A va B: ", D)
```

☞ Kết quả:

```
Hop cua A va B:  {1, 2, 3, 4, 6, 7}
Hop cua A va B:  {1, 2, 3, 4, 6, 7}
```

Phương thức **add()**

Để thêm một phần tử x vào tập hợp A, ta hợp tập A với tập $\{x\}$. Ngoài ra, ta có thể sử dụng phương thức **add()** để thêm một phần tử vào tập A.

Ví dụ 9. Chương trình:

```
A = set()
B = set()
for x in range(5):
    A = A | {x}
    B.add(x)
print(A)
print(B)
```

☞ Kết quả:

```
{0, 1, 2, 3, 4}
{0, 1, 2, 3, 4}
```

c) Phép hiệu

Hiệu của tập hợp A và tập B là một tập hợp gồm tất cả các phần tử thuộc A nhưng không thuộc B.

Kí hiệu: $A - B$

Ví dụ 10. Chương trình:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C =  A - B
print(C)
```

☞ Kết quả:

```
{1, 2}
```

- Phương thức **difference()**

Để tìm hiệu của hai tập, ta có thể sử dụng phương thức **difference()**.

Ví dụ 11. Chương trình:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A.difference(B)
print(C)
```

☞ Kết quả:

```
{1, 2}
```

- Phương thức **discard()**

Để loại bỏ một phần tử x khỏi tập A, ta sử dụng phép hiệu giữa A và $\{x\}$. Ngoài ra ta có thể sử dụng phương thức **discard()**.

Ví dụ 12. Chương trình:

```
A = {1, 2, 3, 4}
A.discard(4)
print(A)
```

☞ Kết quả:

```
{1, 2, 3}
```

d) Phép hiệu đối xứng

Hiệu đối xứng của hai tập hợp A và B là một tập hợp gồm tất cả các phần tử thuộc A hoặc thuộc B nhưng không thuộc cả A và B.

Kí hiệu: A ^ B

Ví dụ 13. Chương trình:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A ^ B
print(C)
```

☞ Kết quả:

```
{1, 2, 5, 6}
```

🖐 Phương thức `symmetric_difference()`

Để tìm hiệu đối xứng của hai tập hợp, ta có thể sử dụng phương thức `symmetric_difference()`

Ví dụ 14. Chương trình:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}
C = A.symmetric_difference(B)
print(C)
```

☞ Kết quả:

```
{1, 2, 5, 6}
```

e) Phép toán `in` và `not in`

🖐 Phép toán `in`

x in A trả về kết quả ***True*** nếu phần tử x thuộc tập ***A***, ngược lại trả về giá trị ***False***.

🖐 Phép toán `not in`

x not in A trả về giá trị ***True*** nếu phần tử x không thuộc tập ***A***, ngược lại trả về giá trị ***False***.

Ví dụ 15. Chương trình:

```
A = {1, 2, 3, 4}
x = 4
print(x in A)
print(x not in A)
```

☞ Kết quả:

```
True
False
```

f) Phép toán đối với tập con

Phép toán so sánh bằng giữa hai tập hợp

Kí hiệu: A == B

Trả về giá trị ***True*** nếu hai tập hợp đó giống nhau, tức là mọi phần tử thuộc tập *A* cũng thuộc tập *B* và mọi phần tử thuộc tập *B* cũng thuộc tập *A*; ngược lại phép so sánh trả về giá trị ***False***.

Phép toán kiểm tra tập con

Kí hiệu: A <= B

Trả về giá trị ***True*** nếu *A* là tập con của *B*, tức là mọi phần tử thuộc tập *A* đều thuộc tập *B*; ngược lại trả về giá trị ***False***.

Phép toán kiểm tra tập con thực sự

Kí hiệu: A < B

Trả về giá trị ***True*** nếu *A* là tập con thực sự của *B*, tức là mọi phần tử thuộc tập *A* đều thuộc tập *B*, nhưng có phần tử thuộc *B* nhưng không thuộc *A*; ngược lại, phép toán trả về giá trị ***False***.

Ví dụ 16. Chương trình:

```
A = {1, 2, 3, 4}
B = {1, 2, 3, 4}
C = {1, 2}
print(A == B)
print(A <= B)
print(C < B)
```

Kết quả:

```
True
True
True
```

g) Phép toán `all()` và `any()`

Trong toán học, ta sử dụng kí hiệu ∀ để nói về mọi phần tử trong một tập hợp và kí hiệu ∃ để nói sự tồn tại của một phần tử trong tập hợp.

Ví dụ, Cho A = {1, 2, 3, 4, 5}, B = $\{\forall x \in A | x : 2\}$ khi đó B = {2, 4}; hay $\exists x \in A | x : 2$. Python cung cấp hai phép toán thực hiện các công việc tương ứng như vậy.

Phép toán `all()`

Câu lệnh **`all(`**[x > 0 ***for x in SetA***])

Có nghĩa tương đương với khẳng định $x > 0, \forall x \in SetA$. Khẳng định này đúng hay sai tùy thuộc vào giá trị của các phần tử trong tập *SetA*.

Ví dụ 17. Chương trình:

```
A = {1, 2, 3, 4}
print(all([x > 0 for x in A]))
print(all([x %2 == 0 for x in A]))
```

Kết quả:

```
True
False
```

Phép toán `any()`

Câu lệnh **any**([x%2 == 0 ***for*** x ***in*** *SetA*]) có nghĩa tương đương với khẳng định $\exists x \in SetA | x : 2$. Khẳng định này đúng hay sai tùy thuộc vào giá trị của các phần tử trong tập *SetA*.

Ví dụ 18. Chương trình:

```
A = {1, 2, 3, 4}
print(any([x < 0 for x in A]))
print(any([x %2 == 0 for x in A]))
```

Kết quả:

```
False
True
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tổng chữ số chung

Viết chương trình nhập hai số tự nhiên a và b. Tính tổng các chữ số chung của a và b. Ví dụ: $a = 1123499$, $b = 1112229$*; có chữ số chung là* $1, 2, 9$*; nên tổng các chữ số chung bằng* $1 + 2 + 9 = 12$. *Nếu có một chữ số cùng xuất hiện nhiều lần trong cả a và b thì chữ số đó chỉ tính một lần.*

Bài tập 2. Nhận phần thưởng

Một lớp học có n học sinh được đánh số thứ tự từ 1 đến n. Nhà trường phát phần thưởng cho các em thành hai đợt.

Đợt 1 gồm k em có số thứ tự $I_1, I_2, .., I_k$.

Đợt 2 gồm t các em có số thứ tự $II_1, II_2, .., II_t$.

Trong mỗi đợt phát thưởng, mỗi em trong danh sách sẽ được nhận một quyển vở.

Yêu cầu: Viết chương trình nhập n và danh sách các em được nhận thưởng đợt 1, đợt 2. Đưa ra danh sách các em được nhận 2 quyển vở, các em nhận được 1 quyển vở, và các em không được nhận quyển vở nào.

Bài tập 3. Danh sách đại diện

Nhập danh sách a gồm n số nguyên $a[0], a[1], .., a[n-1]$. *Tạo danh sách b chỉ gồm các kí tự đại diện trong a, tức là nếu phần tử xuất hiện nhiều lần thì chỉ lấy một phần tử. Các phần tử trong danh sách b được sắp xếp tăng dần.*

Bài tập 4. Học sinh giỏi Toán, Lý, Hóa

Lớp học có n học sinh tham gia kì thi học sinh giỏi cấp tỉnh về ba môn Toán, Lý, Hóa. Các học sinh được đánh số thứ tự từ 1 đến n.

Môn Toán có a học sinh thi, gồm các em có số thứ tự: $t_1, t_2, .., t_a$.

Môn Lý có b học sinh thi, gồm các em có số thứ tự: $l_1, l_2, .., l_b$.

Môn Hóa có c học sinh thi, gồm các em có số thứ tự: $h_1, h_2, .., h_c$.

Yêu cầu: Hãy đưa ra danh sách các em chỉ thi một môn; các em thi hai môn, các em thi cả ba môn.

Ví dụ: n = 10; thi môn Toán gồm các em {1, 2, 3, 7, 8}, thi môn lý {1, 2, 6, 9, 10}, thi môn Hóa {4, 5, 6, 7, 9}.

Thi một môn:

Toán: {3, 8}, lý {10}, Hóa {4, 5}.

Thi hai môn:

Toán, Lý {1, 2}; Lý, Hóa {6, 9}; Toán, Hóa {7}

Thi ba môn: { }, không có em nào.

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tổng chữ số chung

Ý tưởng thuật toán

Ta sử dụng tập hợp *setA* lưa các chữ số của số a, mỗi chữ số xuất hiện một lần trong *setA*, *setB* lưu các chữ số của số b, mỗi chữ số chỉ xuất hiện một lần trong *setB*.

Gọi *setC* là giao của hai tập hợp *setA* và *setB*. Khi đó các phần tử trong *setC* là các chữ số chung của hai số a và b, mỗi chữ số chỉ tính một lần.

Thực hiện cộng các phần tử trong *setC* ta sẽ có kết quả.

Chương trình 1

```
a = input("Nhap so a: ")
b = input("Nhap so b: ")
setA = set(a)
setB = set(b)
setC = setA & setB
sum = 0
for x in setC:
    sum += int(x)
print("Tong chu so chung: ", sum)
```

❖ ***Nhận xét***

Nhận thấy các chữ số chung của a và b chỉ có thể là '0', '1', .., '9'. Vì vậy, ta duyệt lần lượt các chữ số từ chữ số '0' đến chữ số '9'. Với mỗi chữ số, thực hiện kiểm tra xem chữ số đó có thuộc cả hai số a và b không? Nếu thuộc cả a và b thì cộng chữ số này vào tổng.

Chương trình 2

```
a = input("Nhap so a: ")
b = input("Nhap so b: ")
sum = 0
for i in range(10):
    if str(i) in a and str(i) in b:
        sum += i
print("Tong cac chu so chung: ",sum)
```

Bài tập 2. Nhận phần thưởng

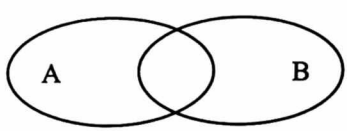
Ý tưởng thuật toán

Ta xem danh sách các học sinh được nhận thưởng đợt 1 là tập A. Danh sách các em được nhận thưởng đợt 2 là tập B. C là tập gồm n số 1, 2, .. , n.

$A = \{I_1, I_2, .. , I_k\}$

$B = \{II_1, II_2, .. , II_t\}$.

$C = \{1, 2, 3, .. , n\}$



Ta nhận thấy, những bạn nhận được hai quyển vở chính là những bạn có số thứ tự thuộc cả tập A và B, tức là thuộc *A&B*.

Những bạn nhận được một cuốn vở là những bạn có số thứ tự thuộc hiệu đối xứng của A và B, tức là thuộc A^B.

Những bạn không nhận được cuốn vở nào thuộc tập $C - (A|B)$.

Chương trình

```
n = int(input("Nhap so hoc sinh"))
k = int(input("Nhap so hoc sinh nhan thuong dot 1:"))
```

```
print("Nhap danh sach hoc sinh nhan thuong dot 1:")
A = {int (input()) for i in range(k)}
t = int(input("Nhap so hoc sinh nhan thuong dot 2:"))
print("Nhap sanh sach hoc sinh nhan thuong dot 2:")
B = {int (input()) for i in range(t)}
C = {i for i in range(1, n+1)}
set1 = A ^ B
set2 = A & B
set0 = C - (A | B)
print("Nhung ban nhan duoc 2 cuon vo:")
print(set2)
print("Nhung ban nhan duoc 1 cuon vo:")
print(set1)
print("Nhung ban khong nhan cuon vo nao:")
print(set0)
```

Bài tập 3. Danh sách đại diện

Ý tưởng thuật toán

Lần lượt cho các phần tử $a[0], a[1], .., a[n-1]$ vào một tập hợp. Các phần tử trùng nhau chỉ được dữ lại một.

Chuyển tập hợp nhận được thành danh sách ta sẽ có một danh sách đại diện.

Chương trình

```
n = int(input("Nhap so phan tu của danh sach: "))
print("Nhap danh sach: ")
a = [int(input()) for i in range(n)]
#Tao tap hop tu cac phan tu trong danh sach a
setA = set(a)
#Tao danh sach tu cac phan tu thuoc tap hop setA
```

```
b = list(setA)
b.sort()
print("Danh sach dai dien:")
print(b)
```

Bài tập 4. Học sinh giỏi Toán, Lý, Hóa

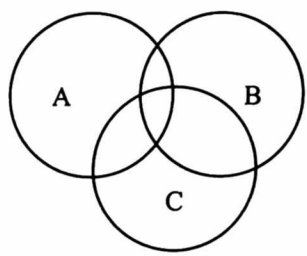
Ý tưởng thuật toán

Ta lưu danh sách các em thi môn Toán trong tập hợp A, môn Lý trong tập hợp B, môn Hóa trong tập C.

$A = \{ t_1, t_2, .., t_a\}$

$B = \{l_1, l_2, .., l_b\}$

$C = \{h_1, h_2, .., h_c\}$



Ta nhận thấy:

Học sinh chỉ thi môn Toán là A – (B | C)

Học sinh chỉ thi môn Lý là B – (A | C)

Học sinh chỉ thi môn Hóa là: C – (A | B)

Số thi cả ba môn là A & B & C.

Số học sinh chỉ thi hai môn gồm:

Chỉ thi hai môn Toán và Lý: A& B – (A&B&C)

Chỉ thi hai môn Lý và Hóa: B&C – (A&B&C)

Chỉ thi hai môn Hóa và Toán: C&A – (A&B&C).

Chương trình

```
a = int(input("Nhap so hoc sinh du thi mon toan:"))
print("Nhap ", a ," hoc sinh thi toan: ")
A = {input() for i in range(a)}
b = int(input("Nhap so hoc sinh du thi mon ly: "))
print("Nhap ", b, " hoc sinh thi ly: ")
B = {input() for i in range(b)}
```

```
c = int(input("Nhap so hoc sinh du thi mon hoa: "))
print("Nhap ",c, " hoc sinh thi hoa: ")
C = {input() for i in range(c)}
Toan  = A - (B | C)
Ly    = B - (A | C)
Hoa   = C - (A | B)
print("Hoc sinh chi thi mon toan: ",Toan)
print("Hoc sinh chi thi mon ly: ", Ly)
print("Hoc sinh chi thi mon hoa: ", Hoa)
TLH  = A & B & C
TL   = (A & B) - TLH
LH   = (B & C) - TLH
HT   = (C & A) - TLH
print("Hoc sinh chi thi toan va ly: ", TL)
print("Hoc sinh chi thi ly va hoa: ", LH)
print("Hoc sinh chi thi hoa va toan: ", HT)
print("Hoc sinh thi toan, ly, hoa: ", TLH)
```

CHỦ ĐỀ 14. KIỂU DỮ LIỆU TỪ ĐIỂN - DICT

A. KIẾN THỨC CƠ BẢN

1. Khái niệm và khai báo

Từ điển (dict) là một danh sách các phần tử, mỗi phần tử gồm hai thành phần, một thành phần được gọi là khóa (*key*), thành phần còn lại gọi là giá trị (*value*) của thành phần đó. Các phần tử trong từ điển có khóa khác nhau, còn giá trị thì có thể bằng nhau. Khóa của mỗi phần tử không thay đổi được, còn giá trị của nó thì có thể thay đổi.

Khai báo:

<biến từ điển> = {*<khóa0>*:*<giátrị0>*, *<khóa1>*:*<giátrị1>*, .. ,*<khóa n-1>*:*<giátrị n-1>*}

Ví dụ 1. *Dict* = {1: 2, 2: "Toan", 3: "Tin"}

Khai báo một biến kiểu dữ liệu từ điển có tên *Dict*, gồm 3 phần tử, có khóa và giá trị tương ứng:

Khóa	Giá trị
1	2
2	"Toan"
3	"Tin"

Các phần tử trong từ điển không được đánh chỉ số như kiểu dữ liệu danh sách, các phần tử trong từ điển được xác định thông qua khóa của chúng. Ta có thể đưa ra giá trị của một phần tử thông qua khóa của nó như sau:

<biến từ điển>[*khóa*].

2. Liệt kê các phần tử thuộc từ điển

a) Liệt kê các khóa

Ví dụ 2. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
```

```
print("Cac khoa cua tu dien:")
for x in Dict:
   print(x, end = ", ")
```

☞ Kết quả:

```
Cac khoa cua tu dien:
1, 2, 3,
```

b) Liệt kê các giá trị

Ví dụ 3. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
print("Cac gia tri cua tu dien:")
for x in Dict:
   print(Dict[x], end = ", ")
```

☞ Kết quả:

```
Cac gia tri cua tu dien:
2, Toan, Tin,
```

Phương thức `values()`

Ngoài ra, ta còn có cách khác để đưa ra các giá trị của các phần tử thuộc từ điển nhờ phương thức **`values()`**

Ví dụ 4. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
print("Cac gia tri cua tu dien:")
for x in Dict.values():
   print(x, end = ", ")
```

☞ Kết quả:

```
Cac gia tri cua tu dien:
2, Toan, Tin,
```

c) Liệt kê cả khóa và giá trị

Phương thức `items()`

Để liệt kê cả khóa và giá trị của các phần tử trong từ điển, ta sử dụng phương thức **`items()`**.

Ví dụ 5. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
print("Khoa va gia tri:")
for x, y in Dict.items():
   print(x,':',y)
```

Kết quả:

```
Khoa va gia tri:
1: 2
2: Toan
3: Tin
```

3. Các phép toán trên kiểu từ điển

a) Lấy số phần tử thuộc từ điển

Hàm ***Len***(*<biến từ điển>*) trả về số phần tử thuộc từ điển *<biến từ điển>*.

Ví dụ 6. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
print("So phan tu cua tu dien: ", Len(Dict))
```

Kết quả:

```
So phan tu cua tu dien:  3
```

b) Thêm một phần tử vào từ điển

<biến từ điển>[<khóa mới>] = <giá trị>

Ví dụ 7. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
Dict[4] = "Ly"
print("Cac phan tu cua tu dien: ")
print(Dict)
```

☞ Kết quả:

```
Cac phan tu cua tu dien:
{1: 2, 2: 'Toan', 3: 'Tin', 4: 'Ly'}
```

c) Thay đổi giá trị của một phần tử

<biến từ điển> [*<khóa>*] = *<giá trị mới>*

Ví dụ 8. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
Dict[4] = "Ly"                          #them moi
Dict[1] = "Ly"                          #thay doi
gia tri
print("Cac phan tu cua tu dien: ")
print(Dict)
```

☞ Kết quả:

```
Cac phan tu cua tu dien:
{1: 'Ly', 2: 'Toan', 3: 'Tin', 4: 'Ly'}
```

d) Xóa một phần tử thuộc từ điển

Có nhiều cách để xóa một phần tử thuộc từ điển. Sau đây là một số cách thực hiện.

- Phương thức **pop()**

<biến từ điển>.**pop**(*<khóa>*)

Xóa phần tử có khóa *<khóa>* khỏi từ điển *<biến từ điển>*.

Ví dụ 9. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
Dict.pop(1)
print("Cac phan tu cua tu dien: ")
print(Dict)
```

☞ Kết quả:

```
Cac phan tu cua tu dien:
{2: 'Toan', 3: 'Tin'}
```

- Phương thức **popitem()**

  <*biến từ điển*>.**popitem()**

Xóa phần tử được đưa vào từ điển lần sau cùng.

Ví dụ 10. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
Dict.popitem()
print("Cac phan tu cua tu dien: ")
print(Dict)
```

☞ Kết quả:

```
Cac phan tu cua tu dien:
{1: 2, 2: 'Toan'}
```

Ví dụ 11. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" }
Dict[5] = "Ly"
Dict[4] = "Hoa"
print("Tu dien truoc khi xoa: ")
print(Dict)
Dict.popitem()
print("Tu dien sau khi xoa:")
print(Dict)
```

☞ Kết quả:

```
Tu dien truoc khi xoa:
{1: 2, 2: 'Toan', 3: 'Tin', 5: 'Ly', 4: 'Hoa'}
Tu dien sau khi xoa:
{1: 2, 2: 'Toan', 3: 'Tin', 5: 'Ly'}
```

- Hàm **del**

• Xóa từng phần tử

del <*biến từ điển*> [<*khóa*>]

Xóa phần tử có khóa <*khóa*> khỏi từ điển <*biến từ điển*>.

Ví dụ 12. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin", 4:"Ly"}
del Dict[2]
print("Tu dien sau khi xoa:")
print(Dict)
```

☞ Kết quả:

```
Tu dien sau khi xoa:
{1: 2, 3: 'Tin', 4: 'Ly'}
```

• Xóa cả từ điển

del <*biến từ điển*>

Xóa cả từ điển <*biến từ điển*>.

Ví dụ 13. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin", 4: "Ly"}
del Dict
print(Dict)
```

Chương trình sẽ báo lỗi ở câu lệnh **print**(Dict) vì lúc này *Dict* không được xác định.

☞ Phương thức **clear()**
<*biến từ điển*>**.clear()**

Xóa tất cả các phần tử thuộc từ điển <*biến từ điển*>.

Ví dụ 14. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin" ,4:"Ly"}
Dict.clear()
print("Tu dien sau khi xoa:")
print(Dict)
```

☞ Kết quả:

```
Tu dien sau khi xoa:
{}
```

Chú ý

Khi xóa một phần tử mà ta truy cập đến khóa không có trong từ điển thì chương trình sẽ báo lỗi **KeyError**.

e) Phép toán in

<khóa> **in** *<biến từ điển>*

Trả về giá trị *True* nếu trong từ điển *<biến từ điển>* tồn tại phần tử có khóa bằng *<khóa>*. Ngược lại, phép toán trả về giá trị *False*.

Ví dụ 15. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin", 4:"Ly"}
if 1 in Dict:
   print("Co phan tu khoa bang 1.")
else:
   print("Khong co phan tu khoa bang 1.")
if 10 in Dict:
   print("Co phan tu khoa bang 10.")
else:
   print("Khong co phan tu khoa bang 10.")
```

☞ Kết quả:

```
Co phan tu khoa bang 1.
Khong co phan tu khoa bang 10.
```

f) Phương thức get()

<biến từ điển>.**get**(*k*)

Trả về giá trị của phần tử có khóa là *k*. Trong trường hợp, từ điểm *<biến từ điển>* không chứa phần tử có khóa *k*, phương thức trả về giá trị **None**.

Ví dụ 16. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin", 4: "Ly"}
print(Dict.get(2))
print(Dict.get(4))
print(Dict.get(5))
```

☞ Kết quả:

```
Toan
Ly
None
```

Dạng tham số

<*biến từ điển*>.**get**(*k*, "giá trị mặc định")

Trả về giá trị của phần tử có khóa là *k*. Trong trường hợp, từ điểm <*biến từ điển*> không chứa phần tử có khóa *k*, phương thức trả về giá trị "giá trị mặc định".

Ví dụ 17. Chương trình:

```
Dict = {1: 2, 2: "Toan", 3: "Tin", 4: "Ly"}
print(Dict.get(2, "Khong ton tai"))
print(Dict.get(4, "Khong ton tai"))
print(Dict.get(5, "Khong ton tai"))
```

☞ Kết quả:

```
Toan
Ly
Khong ton tai
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tạo từ điển

Cho hai danh sánh, danh sách list1 gồm n số khác nhau [a_0, a_1, .., a_{n-1}] và danh sách list2 gồm n tên [$name_0, name_1, .., name_{n-1}$]. Hãy tạo ra một từ điển mà các phần tử có dạng {a_i: $name_i$}, sau đó in nội dung từ điển dưới dạng: <khóa>: <giá trị>.

Bài tập 2. Thống kê điểm thi

Viết chương trình nhập thông tin của học sinh gồm: Điểm, tên và lưu các thông tin này trong một từ điển. Điểm thuộc {0, 1, 2, .., 10}. Sau đó thống kê những bạn được điểm 10, điểm 9, .. , điểm 1, điểm 0.

Bài tập 3. Tra cứu điểm thi

Cho một từ điển chứa thông tin của các thí sinh, mỗi phần tử có 3 thông tin:

- Số báo danh (khóa)

- Họ và tên

- Điểm thi

Chương trình cho phép nhập một số báo danh, nếu tồn tại thí sinh có số báo danh tương ứng thì đưa ra họ và tên cùng điểm. Nếu không có thí sinh trùng với số báo danh được nhập, thông báo không có thí sinh.

Bài tập 4. Cặp số hạng có giá trị liên tiếp

Cho dãy số nguyên $a[0], a[1], .., a[n-1]$. *Tính số cặp chỉ số* (i, j) *sao cho:* $0 \leq i < j \leq n-1$ *và* $a[i] + 1 = a[j]$.

Ví dụ: dãy, $a_1 = 1$, $a_2 = 2$, $a_3 = 3$, $a_4 = 2$, $a_5 = 4$. *Ta có 4 cặp chỉ số:* $(1, 2), (1, 4), (2, 3), (3, 5)$.

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tạo từ điển

Ý tưởng thuật toán

- Nhập *n* là số phần tử.
- Nhập danh sách *list1* = $[a_0, a_1, .., a_{n-1}]$.
- Nhập danh sách list2 = $[name_0, name_1, .., name_{n-1}]$.
- Tạo một từ điển rỗng *dict* = { }.
- Lần lượt thêm vào từ điển dạng *dict*$[a_i]$ = $name_i$.
- In nội dung danh sách gồm khóa và giá trị.

Chương trình

```
n = int(input("Nhap so phan tu n = "))
print("Nhap danh sach gom ",n," so khac nhau: ")
list1 = [int(input()) for i in range(n)]
```

```
print("Nhap danh sach gom ",n, " ten: ")
list2 = [input() for i in range(n)]
dict = {}
for i in range(n):
   dict[list1[i]] = list2[i]
print("Noi dung tu dien:")
for x, y in dict.items():
   print(x,": ",y)
```

Bài tập 2. Thống kê điểm thi

Ý tưởng thuật toán

Để tạo từ điển, ta cần thông tin để tạo khóa. Nhận thấy có thể sử dụng điểm để tạo khóa. Như vậy khóa có giá trị: 0, 1, 2, .. , 10. Ứng với mỗi khóa i ($i = 0,1,..,10$), giá trị là một danh sách các tên của học sinh có điểm là i.

Khi thống kê những bạn được điểm i ($i = 0,1,..,10$), ta đưa ra giá trị của phần tử có khóa là i trong từ điển.

Chương trình

```
n = int(input("Nhap so hoc sinh n = "))
#Tao mot tu dien co 11 khoa
#Gia tri cua cac phan tu la danh sach rong
dict = {}
for i in range(11):
   dict[i] = []
print("Nhap ten va diem cua ",n," hoc sinh:")
for i in range(n):
   name = input("Ten: ")
   mark = int(input("Diem: "))
   dict[mark].append(name)
```

```
print("Thong ke:")
for i in range(11):
    print("Hoc sinh duoc diem ", i,":")
    print(dict[i])
```

Bài tập 3. Tra cứu điểm thi

Ý tưởng thuật toán

Ta sử dụng một danh sách ***val*** để lưu phần giá trị của mỗi phần tử trong từ điển. Trong đó ***val*[0]** lưu họ và tên, ***val*[1]** lưu điểm thi.

Để tra cứu thông tin của học sinh thông qua khóa <*key*>, ta sử dụng phương thức **get**(<key>, "Khong co thi sinh").

Chương trình

```
print("Nhap du lieu cho tu dien")
n = int(input("Nhap so hoc sinh n = "))
print("Nhap SBD, Ten va Diem cua ",n," hoc sinh:")
#Tao mot tu dien rong de chua thong tin nhap vao
dict = {}
for i in range(n):
    SBD   = input("Nhap SBD: ")
    Ten   = input("Nhap Ho va ten: ")
    Diem  = int(input("Nhap diem: "))
    dict[SBD] = [Ten, Diem]
print("Tra cuu:")
SBD = input("Nhap SBD de tra cuu: ")
print(dict.get(SBD,"Khong co thi sinh"))
```

Bài tập 4. Cặp số hạng có giá trị liên tiếp

Ý tưởng thuật toán

Thuật toán 1

Duyệt tất cả các cặp chỉ số (i,j), ứng với mỗi cặp chỉ số (i,j), nếu thỏa mãn $a_i + 1 = a_j$ thì tăng số cặp cần tính lên 1.

❖ *Nhận xét*

Nhận thấy, thuật toán này rõ ràng và dễ cài đặt nhưng số phép tính cần thực hiện khoảng n^2. Thuật toán sau đây sử dụng cấu trúc từ điển cho số lượng phép tính cần thực hiện $nlog_2n$.

Thuật toán 2

Lần lượt xét các số hạng $a[0], a[1], .., a[n-1]$. Với mỗi số hạng $a[j]$, ta cần tính xem có bao nhiêu số hạng trước đó ($a[i]$ với $0 \leq i < j$) mà $a[i] + 1 = a[j]$. Có bao nhiêu số hạng như vậy sẽ có bấy nhiêu cặp ứng với chỉ số j đang xét. Để thực hiện điều này, ta sử dụng một từ điển mà phần tử của nó có dạng: $\{a[t]: s\}$, trong đó s là số lần xuất hiện của số hạng $a[t]$ trong dãy.

Việc xây dựng từ điển, ta tiến hành tuần tự với quá trình xét các số hạng $a[j]$. Ban đầu, ta tạo từ điển rỗng ($dict$).

Với số hạng $a[j]$, số các số hạng $a[i]$, $(0 \leq i < j)$ mà $a[i] + 1 = a[j]$ chính là $dict[a[j] - 1]$. Chú ý là thao tác truy xuất giá trị này có số lượng phép tính là log_2n. Sau đó cập nhật số lần xuất hiện của số $a[j]$ bằng cách thực hiện $dict[a[j]] = dict[a[j]] + 1$.

Chương trình 1

```
n = int(input("Nhap so phan tu cua day: "))
print("Nhap ",n," phan tu: ")
a = [int(input()) for i in range(n) ]
kq = 0
for i in range(n-1):
   for j in range(i+1, n):
      if a[i] + 1 == a[j]:
         kq = kq + 1
print("So cap co gia tri lien tiep: ", kq)
```

Chương trình 2

```
n = int(input("Nhap so phan tu cua day: "))
print("Nhap ",n," phan tu: ")
```

```
a = [int(input()) for i in range(n) ]
kq = 0
dict = {}
for j in range(n):
   if dict.get(a[j] - 1) != None:
      kq = kq + dict[a[j] - 1]
   if dict.get(a[j]) == None:
      dict[a[j]] = 1
   else:
      dict[a[j]] = dict[a[j]] + 1
print("So cap so hang co gia tri lien tiep: ",kq)
```

CHỦ ĐỀ 15. KIỂU DỮ LIỆU TỆP

A. KIẾN THỨC CƠ BẢN

Ở các bài học trước, dữ liệu được đưa từ bài phím để xử lý và kết quả được ghi ra màn hình. Ở bài học này, ta sẽ đề cập đến dữ liệu được đọc từ tệp vào để xử lý và kết quả cũng được ghi ra tệp. Tệp là một tập hợp các thông tin được lưu ở bộ nhớ ngoài, nó tạo thành đơn vị lưu trữ và không mất đi khi máy tính mất nguồn điện. Ở đây ta chỉ đề cập đến tệp văn bản, nội dung của chúng được biểu diễn dưới dạng các kí tự. Các kí tự trong tệp được tổ chức theo từng dòng, mỗi dòng có kí tự hết dòng (EOL). Hệ điều hành không cho các chương trình làm trực tiếp với tệp dữ liệu mà thông qua biến tệp. Điều này giúp cho an toàn về dữ liệu trong tệp.

1. Mở tệp để ghi dữ liệu

a) Khai báo biến tệp

<biến tệp> = **open(***<tên tệp>***, 'w')**

Khi thực hiện câu lệnh này, chương trình sẽ tạo ra một tệp văn bản với tên tương ứng để ghi dữ liệu vào. Ban đầu tệp chưa chứa dữ liệu nào. Trong trường hợp tệp đã tồn tại thì dữ liệu cũ sẽ bị xóa hết. Tên tệp có thể gồm cả đường dẫn.

Ví dụ 1. *f1* = **open**("Vidu.txt",'w')

f2 = **open**("C:/python/Vidu.txt",'w')

b) Ghi dữ liệu vào tệp

- Câu lệnh **write()**

<biến tệp>.**write(***<chuỗi kí tự>***)**

☞ Ghi một chuỗi kí tự vào tệp.

Ví dụ 2. Chương trình:

```
f = open("Vidu.txt",'w')
f.write("Lap trinh Python")
f.write(" rat thu vi")
f.close()                          #dong tep
```

☞ Kết quả:

Ví dụ 3. Chương trình:

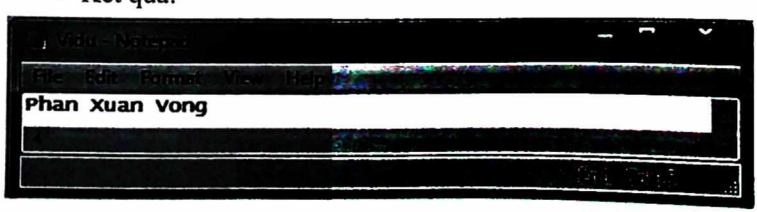
```
f = open("Vidu.txt",'w')
f.write("Lap trinh Python\n")
f.write("Rat thu vi")
f.close()
```

☞ Kết quả:

Ví dụ 4. Chương trình:

```
f = open("Vidu.txt",'w')
ho = "Phan Xuan"
ten = " Vong"
f.write(ho)
f.write(ten)
f.close()
```

☞ Kết quả:



❖*Nhận xét*

Câu lệnh **write()** *sẽ ghi* ***một*** *dãy các kí tự vào tệp và* ***không*** *xuống dòng. Để xuống dòng, cần đưa thêm kí tự xuống dòng '\n' vào.*

Chương trình sau sẽ báo lỗi:

```
f = open("Vidu.txt",'w')
lis = ["a", "b", "c"]
f.write(lis)
f.close()
```

```
TypeError: write() argument must be str, not list
```

Để khắc phục điều này, ta sử dụng lệnh **writelines().**

🖙 *Câu lệnh* **writelines()**

<*biến tệp*>.**writelines**(<*một đối tượng*>)

☛ *Ghi nội dung một đối tượng vào tệp. Nội dung của đối tượng phải là các dãy kí tự.*

Ví dụ 5. Chương trình:

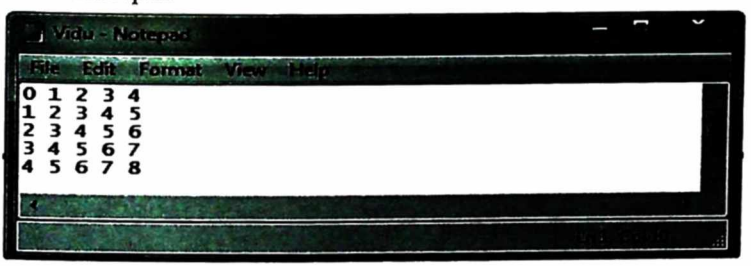
```
f = open("Vidu.txt",'w')
lis = ["a", "b", "c"]
f.writelines(lis)
f.writelines('\n')
for x in lis:
    f.writelines(x + '\n')
f.close()
```

☛ Kết quả:

Ví dụ 6. Chương trình:

```
f = open("Vidu.txt",'w')
for i in range(5):
    for j in range(5):
        f.write(str(i+j)+ ' ')
    f.write('\n')
f.close()
```

☞ Kết quả:



❖ ***Chú ý***

Chương trình sau sẽ bị lỗi vì biến i được xem là kiểu số. Khắc phục điều này, cần chuyển kiểu số sang kiểu chuỗi bằng cách **str**(*i*).

- *Câu lệnh sai*:

```
for i in range(5):
   f.write(i)
```

- *Câu lệnh đúng*:

```
for i in range(5):
   f.write(str(i))
```

2. Mở tệp để đọc dữ liệu

a) Khai báo biến tệp

<*biến tệp*> = **open**(<*tên tệp*>, 'r')

Khi thực hiện câu lệnh này, nếu không tồn tại tệp, chương trình sẽ báo lỗi không tìm thấy tệp: **FileNotFoundError.**

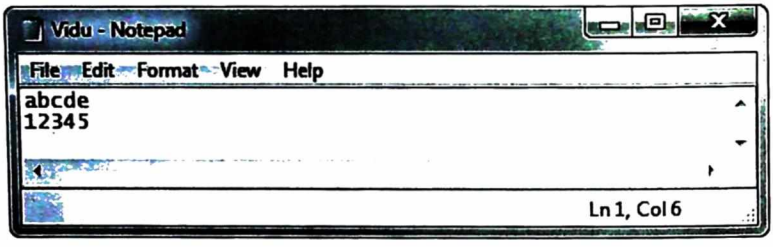
b) Đọc dữ liệu

Câu lệnh `read()`

<biến tệp>`.read()`

☛ Đọc tất cả các kí tự trong tệp (kể cả kí tự xuống dòng) và lưu chúng thành một chuỗi kí tự.

Ví dụ 7. Cho tệp có nội dung:



Chương trình:

```
f = open("Vidu.txt",'r')
read_all = f.read()
print(read_all)
print("Do dai: ", len(read_all))
f.close()
```

☛ Kết quả:

```
abcde
12345

Do dai:  12
```

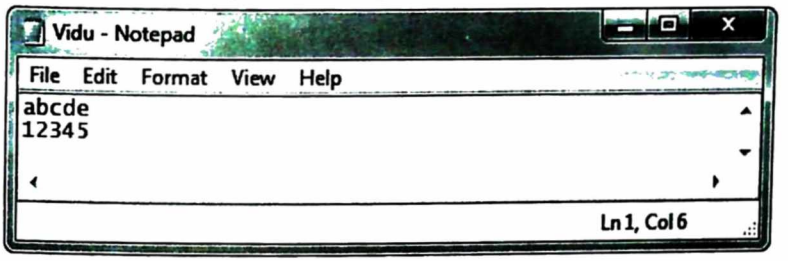
Ta nhận thấy, kết quả đọc được lưu trong biến chuỗi `read_all`, độ dài bằng 12 là vì tính cả hai kí tự xuống dòng khi đọc tệp.

Dạng tham số

<biến tệp>`.read(n)`

☛ Đọc *n* byte tính từ vị trí con trỏ tệp. Khi các kí tự được mã hóa theo bảng mã **ASCII** thì mỗi kí tự được tính là 1 byte.

Ví dụ 8. Cho tệp có nội dung:



Chương trình:

```
f = open("Vidu.txt",'r')
read_byte = f.read(3)
print(read_byte)
read_byte = f.read(2)
print(read_byte)
f.close()
```

☞ Kết quả:

```
abc
de
```

Câu lệnh `readline()`

`<`*biến tệp*`>.readline()`

☞ Đọc từng dòng của tệp văn bản.

Ví dụ 9. Chương trình:

```
f = open("Vidu.txt",'r')
read_line = f.readline()
print(read_line)
read_line = f.readline()
print(read_line)
f.close()
```

☞ Kết quả:

```
abcde

12345
```

• ***Chú ý***

Câu lệnh **readline()** *sẽ đọc từng dòng gồm cả kí tự xuống dòng, do vậy khi in ra, ta sẽ thấy có một dòng trống xuất hiện.*

- *Câu lệnh* **readlines()**

  <*biến tệp*>**.readlines()**

☞ *Đọc cả tệp và lưu chúng thành một danh sách, mỗi phần tử của danh sách là một dòng (gồm cả kí tự xuống dòng).*

Ví dụ 10. Chương trình:

```
f = open("Vidu.txt",'r')
read_list = f.readlines()
print(read_list)
f.close()
```

☞ Kết quả:

```
['abcde\n', '12345\n']
```

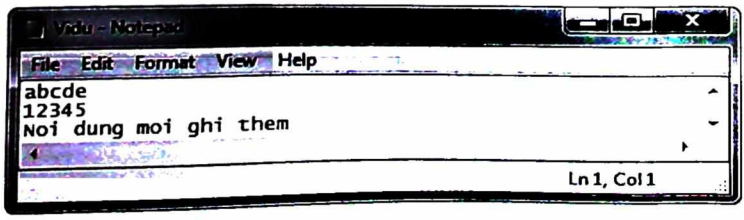
3. Mở tệp để ghi thêm dữ liệu

<*biến tệp*> = **open(**<*tên tệp*>**,'a')**

Ví dụ 11. Chương trình:

```
f = open("Vidu.txt",'a')
f.write("Noi dung moi ghi them")
f.close()
```

☞ Kết quả:



❖ *Chú ý*

Khi mở tệp để ghi thêm nội dung vào, nếu tệp không tồn tại, chương trình sẽ tự động tạo ra một tệp chưa có nội dung để ghi nội dung mới vào.

4. Đóng tệp

Khi thực hiện xong việc đọc hay ghi dữ liệu từ tệp, để đảm bảo dữ liệu cũng như chuyển đổi chế độ làm việc (từ ghi sang đọc, ...) ta cần thực hiện thao tác đóng tệp.

<*biến tệp*>.close()

5. Một số cách khác để ghi dữ liệu, đọc dữ liệu từ tệp

Câu lệnh with

- Ghi tệp

```
with open(<tên tệp>,'w') as <biến tệp>:
    <Khối lệnh làm việc với tệp>
```

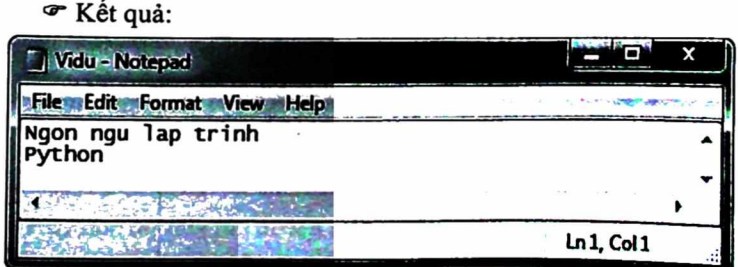
- Đọc tệp

```
with open(<tên tệp>,'r') as <biến tệp>:
    <Khối lệnh làm việc với tệp>
```

Ví dụ 12. Chương trình:

```
with open("C:/Vidu.txt",'w') as f:
    f.write("Ngon ngu lap trinh\n")
    f.write("Python")
```

☞ Kết quả:



Ví dụ 13. Chương trình:

```
with open("C:/Vidu.txt",'r')  as f:
    st = f.read()
print(st)
```

☞ Kết quả:

```
Ngon ngu lap trinh
Python
```

Hằng file và câu lệnh print()

Ví dụ 14. Chương trình:

```
f = open("C:/Vidu.txt",'w')
for i in range(5):
    print(i, file = f)
for i in range(5):
    print(i, end = ' ',file = f)
```

☞ Kết quả:

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tổng các số hạng lẻ

Cho dãy số gồm các số nguyên dương được ghi trong tệp Dayso.txt. Hãy đọc các số hạng của dãy số này và tính tổng các số hạng lẻ trong dãy. Các số hạng có thể được ghi trên nhiều dòng.

Ví dụ:

Dayso.txt
121 33 10
10 20 1

Tổng các số hạng lẻ trong tệp là: 121 + 33 + 1 = 155.

Bài tập 2. Phân loại dữ liệu

Cho tệp dữ liệu có tên Data.txt gồm các kí tự La-tinh và các kí tự số, các kí tự có thể ghi trên nhiều dòng. Hãy đọc dữ liệu từ tệp Data.txt và phân loại ra dạng kí tự La-tinh và kí tự số. Kí tự La-tinh ghi vào tệp Latinh.txt, kí tự chữ số ghi vào tệp Chuso.txt. Các kí tự được ghi lần lượt theo thứ tự xuất hiện trong tệp Data.txt.

Ví dụ:

Data.txt	Latinh.txt	Chuso.txt
Abc12bb	Abcbb	12
12bbbc123	Bbbc	12123
Abb	Abb	1234
1234	Abc	12
12abc		

Bài tập 3. Ráp phách

Cho ba tệp dữ liệu Sbd_Ph.txt, Sbd_Ten.txt và Ph_Diem.txt.

- Tệp Sbd_Ph.txt – Gồm số báo danh và số phách.

Chứa thông tin về số báo danh và số phách.

Mỗi dòng ghi hai số nguyên dương a và b tương ứng là số báo danh và số phách của một thí sinh.

- Tệp Sbd_Ten.txt – Gồm số báo danh và họ tên.

Chứa thông tin về số báo danh và họ tên.

Mỗi dòng ghi số nguyên dương a và chuỗi st tương ứng là số báo danh và họ tên của thí sinh.

- Tệp Ph_Diem.txt – Gồm số phách và điểm.

Chứa thông tin về số phách và điểm.

Mỗi dòng ghi hai số nguyên a và b tương ứng là số phách và điểm của một thí sinh.

Hãy đọc dữ liệu từ các tệp đã cho để ráp thông tin:

Số báo danh, họ và tên, điểm của một thí sinh.

Hãy sắp xếp danh sách thí sinh này theo điểm giảm dần và ghi dữ liệu này ra tập Ketqua.txt. Mỗi dòng của tệp ghi thông tin:

Số báo danh, họ và tên, điểm của một thí sinh.

Dữ liệu trong các tệp Sbd_Ph.txt, Sbd_Ten.txt và Ph_Diem.txt được xem là chính xác.

Ví dụ:

Sbd_Ph.txt	Sbd_Ten.txt	Ph_Diem.txt
1 10	1 Nguyen Van An	10 9
2 11	2 Tran Van Binh	16 8
4 12	3 Le Van Manh	11 7
3 16	4 Nguyen Thi Lan	12 10

Ta có:

Ketqua.txt
4: Nguyen Thi Lan: 10
1: Nguyen Van An: 9
3: Le Van Manh: 8
2: Tran Van Binh: 7

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tổng các số hạng lẻ

Ý tưởng thuật toán

- Mở tệp Dayso.txt để đọc dữ liệu.

- Đọc dữ liệu của tệp theo từng dòng và lưu chúng dạng một chuỗi kí tự.

- Thực hiện tách các số hạng trong chuỗi, mỗi số hạng tách được, nếu là số hạng lẻ thì cộng chúng vào tổng kết quả.

Để tách các số hạng trong chuỗi, ta sử dụng phương thức tách **split()**, danh sách ta nhận được gồm các số hạng, các kí tự trống, kí tự xuống dòng.

Chương trình

```
f = open("Dayso.txt",'r')
list_lines = f.readlines();
f.close()
kq = 0
for st in list_lines:
   list = st.split()
   for x in list:
      if x != '\n' and x != '':
         y = int(x)
         if y%2  == 1:
            kq = kq + y
print("Tong cac so hang le: ",kq)
```

Bài tập 2. Phân loại dữ liệu

Ý tưởng thuật toán

Mở tệp Data.txt để đọc dữ liệu, tệp Latinh.txt và Chuso.txt để ghi dữ liệu.

Đọc từng dòng trong tệp Data.txt và xử lý đối với từng dòng đó.

Với mỗi dòng:

Tạo hai chuỗi *alpha*, *digit* để lưu các kí tự La-tinh và kí tự chữ số trong dòng đang xử lý (bắt đầu xử lý mỗi dòng, hai chuỗi này được gán là chuỗi rỗng).

Xét từng kí tự trên một dòng, nếu là kí tự La-tinh thì lưu vào chuỗi *alpha*, nếu là kí tự chữ số thì lưu vào chuỗi *digit*.

Nếu chuỗi *alpha* khác rỗng thì ghi vào tệp Latinh.txt.

Nếu chuỗi *digit* khác rỗng thì ghi vào tệp Chuso.txt.

Chương trình

```
f       = open("Data.txt",'r')
so      = open("Chuso.txt", 'w')
latinh  = open("Latinh.txt",'w')
read_lines = f.readlines()
for st in read_lines:
   alpha = ""
   digit = ""
   for x in st:
      if x.isdigit():
         digit = digit + x
      if x.isalpha():
         alpha = alpha + x
   if alpha > "":
      latinh.write(alpha+'\n')
   if digit > "":
      so.write(digit+'\n')
f.close()
so.close()
latinh.close()
```

Bài tập 3. Ráp phách

Ý tưởng thuật toán

- Đọc dữ liệu từ tệp Sbd_Ten.txt theo từng dòng, với mỗi dòng, tách thành hai thành phần: Số báo danh <Sbd> và Họ tên <Ten>.
  Hai phần này được phân cách nhau bởi kí tự trắng đầu tiên.
  Lưu dữ liệu trong một từ điển **sbd_ten_dict** dạng {<Sbd>: <Ten>}.
- Đọc dữ liệu từ tệp Ph_Diem.txt theo từng dòng, với mỗi dòng, tách thành hai phần: Phách <phach> và điểm <diem>.

Hai phần này được phân cách nhau bởi kí tự trắng ở giữa.

Lưu dữ liệu trong một từ điển **ph_diem_dict** dạng {<phach>: <diem>}.

- Đọc dữ liệu từ tệp Sbd_ph.txt theo từng dòng và xử lý ráp phách:
  Với mỗi dòng tách thành hai phần: Số báo danh **<Sbd>** và phần phách **<phach>**.
  Ứng với mỗi cặp <Sbd> và <phach> của một thí sinh, thì tên và điểm của thí sinh đó là:
  ten = sbd_ten_dict[Sbd]
  diem =ph_diem_dict[phach]

Ta lưu thông tin của một thí sinh dưới dạng một bộ (diem, ten, sbd) và đưa bộ này vào trong một danh sách kết quả: **list_kq**.

Sắp xếp danh sách **list_kq** theo thứ tự giảm dần. Chú ý là để sắp xếp theo thứ tự giảm dần, thành phần **diem** phải chuyển sang kiểu dữ liệu **int**.

Chương trình

```
#Mo cac tep
sbd_ph_file      = open("Sbd_Ph.txt",'r')
sbd_ten_file  = open("Sbd_Ten.txt",'r')
ph_diem_file  = open("Ph_Diem.txt",'r')
ketqua_file      = open("Ketqua.txt",'w')
#Doc du lieu tu cac tep
sbd_ph_lines     = sbd_ph_file.readlines()
sbd_ten_lines    = sbd_ten_file.readlines()
ph_diem_lines    = ph_diem_file.readlines()
#Tao tu dien luu sbd - ten
sbd_ten_dict  = {}
#Xet tung dong trong tep sbd_ten.txt
for x in sbd_ten_lines:
   x.strip()
   list  = x.split(' ',1)
```

```
    sbd   = list[0].strip()
    ten   = list[1].strip()
    sbd_ten_dict[sbd] = ten
#Tao tu dien luu ph_diem
ph_diem_dict  = {}
#Xet tung dong trong tep ph_diem.txt
for x in ph_diem_lines:
    x.strip()
    list  = x.split()
    phach    = list[0].strip()
    diem  = list[1].strip()
    ph_diem_dict[phach] = diem
#Tien hanh rap phach
list_kq = []
#Xet tung dong trong tep sbd_ph.txt
for x in sbd_ph_lines:
    x.strip()
    list  = x.split()
    sbd   = list[0].strip()
    phach    = list[1].strip()
    #Chuyen sang kieu du lieu int de sap xep
    diem  = int(ph_diem_dict[phach])
    ten   = sbd_ten_dict[sbd]
    list_kq.append((diem, ten, sbd))
#Sap xep danh sach hoc sinh theo diem giam dan
list_kq.sort(reverse =  True)
#Ghi ket qua ra tep Ketqua.txt
```

```
for x in list_kq:
    line = x[2]+': ' + x[1] + ': ' + str(x[0])
    ketqua_file.write(line)
    ketqua_file.write('\n')
#Dong cac tep da mo
sbd_ph_file.close()
sbd_ten_file.close()
ph_diem_file.close()
ketqua_file.close()
```

Chương 3. MỘT SỐ CHỦ ĐỀ PYTHON NÂNG CAO

CHỦ ĐỀ 16. CHƯƠNG TRÌNH CON

A. KIẾN THỨC CƠ BẢN

1. Khái niệm

Chương trình con (*Subprogram*) là một dãy các lệnh mô tả một số thao tác nhất định và có thể được thực hiện (được gọi) từ nhiều vị trí trong chương trình.

2. Cấu trúc chương trình con

def *<tên chương trình con> (<tham số>):*

[khối lệnh]

Ví dụ 1. Chương trình:

```
def  Tamgiac():
   print("*")
   print("**")
   print("***")
   print("****")
#----------------- het chuong trinh con
Tamgiac()          # goi chuong trinh con
```

☛ Kết quả:

```
*
**
***
****
```

Ví dụ 2. Chương trình:

```
def  In_Python(n):
   for i in range(n):
      print("Python")
# ----------------------- het chuong trinh con
n = 3
In_Python(n)              #goi chuong trinh con
```

☞ Kết quả:

```
Python
Python
Python
```

Ví dụ 3. Chương trình:

```
def  Tong(a, b):
   c = a + b
   return c
#----------------- het chuong trinh con
a = 4
b = 6
print("Tong = ", Tong(a, b))
print("Tong = ", Tong(10, 20))
```

☞ Kết quả:

```
Tong =  10
Tong =  30
```

❖ ***Nhận xét***

Nếu chương trình con không trả về một giá trị cụ thể (ví dụ 1) thì ta không cần viết câu lệnh **return**. *Trong những trường hợp chương trình con cần trả về một giá trị sau khi tính toán thì sử dụng câu lệnh* **return** *<giá trị> để trả về <giá trị> đó qua tên của chương trình con.*

Khi thực hiện câu lệnh **return**, *chương trình con sẽ dừng ngay lập tức và các câu lệnh ở phía sau trong chương trình con sẽ không được thực hiện.*

Ví dụ 4. Chương trình:

```
def  In():
    print("Lap trinh Python")
    return
    print("Cau lenh khong duoc thuc hien")
In()
```

☞ Kết quả:

```
Lap trinh Python
```

3. Tham số mặc định

Ở một số chương trình trong các bài học trên, ta đã bắt gặp những hàm có tham số mặc định, như hàm *print*(). Python cũng cho phép người lập trình thiết kế chương trình con với các tham số mặc định. Khi thực hiện chương trình con, nếu không truyền giá trị cho tham số thì chương trình sẽ hiểu giá trị lúc này là giá trị mặc định, trong trường hợp có truyền giá trị thì tham số sẽ nhận giá trị đó.

Cách viết tham số mặc định

def *<tên chương trình con> (<tham số> = <giá trị mặc định>):*
[khối lệnh]

Ví dụ 5. Chương trình:

```
def  In_Python(n = 2):
    for i in range(n):
        print("Python")
In_Python()#goi chuong trinh con voi tham so
mac dinh
In_Python(3) #goi chuong trinh con voi tham so bang 3
```

☞ Kết quả:

```
Python
Python
Python
Python
Python
```

❖ ***Chú ý***

Chương trình con có thể có nhiều tham số, các tham số được phân cách nhau bởi dấu phẩy. Các tham số đều có thể sử dụng giá trị mặc định.

Ví dụ 6. Chương trình:

```
def  Tong(a = 0, b = 0, c = 0):
        s = a + 2*b + 3*c
        return s
print("Tong: ",Tong())    # a = 0, b = 0, c = 0
print("Tong: ",Tong(1))   # a = 1, b = 0, c = 0
print("Tong: ",Tong(1, 2)) # a = 1, b = 2, c = 0
print("Tong: ",Tong(1, 2, 3)) # a = 1, b = 2, c = 3
```

☞ Kết quả:

```
Tong:  0
Tong:  1
Tong:  5
Tong:  14
```

4. Danh sách các tham số

Một sự linh hoạt trong thiết kế về tham số trong chương trình con mà Python cung cấp đó là danh sách các tham số. Cũng một chương trình con đó, nhưng mỗi lần gọi thực hiện có thể khác nhau về cả số lượng tham số và kiểu tham số.

def *<tên chương trình con> (*<tham số>):*

[khối lệnh]

Ví dụ 7. Chương trình:

```
def  Tong(*a):
    s = 0
    for x in a:
        s += x
    return s
print("Tong: ",Tong())
print("Tong: ",Tong(1))
print("Tong: ",Tong(1, 2))
print("Tong: ",Tong(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
```

☞ Kết quả:

```
Tong:  0
Tong:  1
Tong:  3
Tong:  55
```

5. Biến toàn cục và biến cục bộ

- **Biến cục bộ** (*local variables*)

Biến cục bộ là biến được định nghĩa trong một chương trình con. Phạm vi sử dụng của chúng chỉ trong chương trình con đó.

- **Biến toàn cục** (*global variables*)

Biến toàn cục là các biến được định nghĩa ngoài chương trình. Phạm vi sử dụng ở mọi vị trí trong chương trình (cả trong chương trình con và ngoài chương trình con).

Ví dụ 8. Chương trình:

```
n = 10                  #global variable
def Tong():
   i = 0                #local variable
   s = 0                #local variable
   while i <= n:
      s = s + i
      i = i + 1
   return s
print("Tong: ",Tong())
```

☞ Kết quả:

```
Tong:  55
```

Ví dụ 9. Chương trình:

```
n = 10                  #global variable
def Tong():
   i = 0                #local variable
   s = 0                #local variable
   n = 20               #local variable
   while i <= n:
      s = s + i
      i = i + 1
   return s
print("Tong: ",Tong())
print("Gia tri n: ",n)
```

☞ Kết quả:

```
Tong: 210
Gia tri n: 10
```

❖ *Nhận xét*

Trong ví dụ 8, giá trị biến toàn cục n được sử dụng trong chương trình con **Tong()**.

Trong ví dụ 9, có hai biến cùng tên là n (Một biến cục bộ và một biến toàn cục). Ta thấy trong chương trình con **Tong()**, *giá trị của biến cục bộ được sử dụng, nhưng ở ngoài chương trình con, câu lệnh:* **print**("Gia tri n: ",n) *thì biến n toàn cục được sử dụng.*

Như vậy, nếu hai biến cùng tên, một biến toàn cục, một biến cục bộ, thì chương trình con chứa biến cục bộ sẽ sử dụng biến cục bộ đó.

Ví dụ 10. Chương trình:

```
def Tong():
    n = 10              #local variable
    s = 0               #local variable
    i = 0               #local variable
    while i <= n:
        s = s + i
        i = i + 1
    return s
print("Tong: ",Tong())
print("Gia tri n: ", n)
```

Khi thực hiện chương trình sẽ thông báo lỗi ở câu lệnh:

```
print("Gia tri n: ", n)
NameError: name 'n' is not defined
```

Lí do câu lệnh bị lỗi: biến *n* là biến cục bộ, chỉ được sử dụng trong chương trình con chứa biến đó. Khi ra khỏi chương trình con, biến *n* xem như chưa được khai báo.

Ví dụ 11. Chương trình:

```
n = 10                  #global variable
s = 0                   #global variable
```

```
def Tong():
    i = 0              #local variable
    while i <= n:
        s = s + i
        i = i + 1
    return s
print("Tong: ",Tong())
```

Chương trình sẽ báo lỗi ở câu lệnh: s = s + i. `UnboundLocalError: local variable 's' referenced before assignment`

Có nghĩa là, biến *s* được xem là biến cục bộ và chưa được gán giá trị khi tham chiếu đến *s*. Để khắc phục vấn đề này, ta sử dụng từ khóa **global** trước biến *s* được khai báo trong chương trình con. Và biến *s* được sử dụng như biến toàn cục ở phần dưới chương trình.

Ví dụ 12. Chương trình:

```
s = 0
def Tong():
    global n    #global variable
    global s    #global variable
    i = 0       #local variable
    while i <= n :
        s = s + i
        i = i + 1
    return s
n = 10
print("Tong: ",Tong())
print("Gia tri s: ", s)
```

☞ Kết quả:

```
Tong:  55
Gia tri s:  55
```

6. Chương trình con trong chương trình con

Python cho phép thiết kế chương trình con trong chương trình con. Sau đây là một số ví dụ và lưu ý khi đề cập đến vấn đề này.

Ví dụ 13. Chương trình:

```
def Tong_Chan_Le(*a):
    def  Tong_chan():
        s = 0
        for x in a:
            if x % 2 == 0: s = s + x
        return s
    #---------------het chuong trinh con
Tong_chan()
    def Tong_le():
        s = 0
        for x in a:
            if x%2 != 0: s = s + x
        return s
    #---------------het chuong trinh con
Tong_le()
    print("Tong chan: ",Tong_chan())
    print("Tong le: ",Tong_le())
#---------------------------------
Tong_Chan_Le(1, 2, 3, 4)
```

☛ Kết quả:

```
Tong chan:  6
Tong le:  4
```

Ví dụ 14. Chương trình:

```
def func1():
    s = 1
    def  func2():
        print("in func2 s = ", s)
    #---------------
    func2()
    print("in func1 s = ", s)
#-------------------
func1()
```

☞ Kết quả:

```
in func2 s =  1
in func1 s =  1
```

Trong chương trình con **func1()** có sử dụng một biến cục bộ *s*, nhưng biến cục bộ này có thể sử dụng trong chương trình con **func2()** thuộc chương trình con **func1()**.

Ví dụ 15. Chương trình:

```
def func1():
    s = 1
    def  func2():
        s = s + 1
        print("in func2 s = ", s)
    #---------------
    func2()
    print("in func1 s = ", s)
#-------------------
func1()
```

Khi thực hiện, chương trình sẽ báo lỗi ở câu lệnh **s = s + 1**.

```
UnboundLocalError: local variable 's' referenced before
assignment
```

Điều này có nghĩa là biến *s* là biến cục bộ nên không thể thay đổi giá trị của nó trong chương trình con ở mức tiếp theo (mức trong). Để sử dụng một biến *s* duy nhất trong trường hợp này, ta dùng từ khóa **nonlocal** trước biến *s*. Khi đó biến s có phạm vi sử dụng ở mức rộng hơn nhưng không phải là biến toàn cục.

Ví dụ 16. Chương trình:

```
def func1():
    s = 1
    def  func2():
        nonlocal s
        s = s + 1
        print("in func2 s = ", s)
    #---------------
    func2()
    print("in func1 s = ", s)
#-------------------
func1()
```

☛ Kết quả:

```
in func2 s =  2
in func1 s =  2
```

7. Thay đổi giá trị của tham số

Khi thực hiện truyền giá trị cho tham số, các ngôn ngữ lập trình khác như C, C++, .., việc truyền giá trị có tham số ở dạng truyền giá trị (tham trị - *Call by Value*) và truyền địa chỉ ô nhớ chứa biến có giá trị cần truyền (tham biến - *Call by Reference*). Với Python không phân biệt rõ những trường hợp như vậy mà phụ thuộc vào kiểu dữ liệu được sử dụng trong tham số và các câu lệnh tác động lên các biến này. Sau đây là một số ví dụ về sự thay đổi giá trị của tham số.

Ví dụ 17. Chương trình:

```
def Tangthem(a):
    a = a + 10
    return a
a = 10
print(Tangthem(a))
print("Gia tri a: ", a)
```

☞ Kết quả:

```
20
Gia tri a:  10
```

Nhận thấy, giá trị a không thay đổi khi thực hiện xong chương trình con, mặc dù trong thân chương trình con có câu lệnh $a = a + 10$.

Ví dụ 18. Chương trình:

```
def thaydoi(L):
    L.append(1)
    print("Danh sach trong chuong trinh con")
    print(L)
L = [2, 3]
print("Danh sach truoc khi thuc hien chuong
trinh con")
print(L)
thaydoi(L)
print("Danh sach sau khi thuc hien chuong trinh con")
print(L)
```

☞ Kết quả:

```
Danh sach truoc khi thuc hien chuong trinh con
[2, 3]
Danh sach trong chuong trinh con
[2, 3, 1]
```

```
Danh sach sau khi thuc hien chuong trinh con
[2, 3, 1]
```

Ta nhận thấy, danh sách trong chương trình con đã thay đổi v giữ nguyên giá trị đó ra khỏi chương trình con. Điều này có thể lý gi trong kiểu dữ liệu danh sách (set, dict, ..) có phương thức ***append(*** không làm thay địa chỉ ô nhớ chứa giá trị biến đó, do vậy mặc dù câ lệnh được thực hiện trong chương trình con nhưng khi ra khỏi chươn trình con thì vẫn giữ nguyên giá trị của biến.

Ví dụ 19. Chương trình:

```
list1  = [1]
list2  = [1]
print("Dia chi o nho list1 ban dau: ",id(list1))
print("Dia chi o nho list2 ban dau: ",id(list2))
list1.append(2)
list2 = list2 + [2]
print("list1: ", list1)
print("list2: ", list2)
print("Dia chi o nho list1 sau: ",id(list1))
print("Dia chi o nho list2 sau: ",id(list2))
```

☛ Kết quả:

```
Dia chi o nho list1 ban dau:  10175952
Dia chi o nho list2 ban dau:  10177112
list1:  [1, 2]
list2:  [1, 2]
Dia chi o nho list1 sau:  10175952
Dia chi o nho list2 sau:  10535360
```

Nhận thấy địa chỉ ô nhớ của giá trị biến *list*1 không thay đổi, đ chỉ biến *list*2 đã thay đổi khi thực hiện thay đổi giá trị của *list*2.

8. Tham số là chương trình con

Python cũng cung cấp cho người lập trình kiểu dữ liệu chương trình con. Python coi chương trình con là một dạng kiểu dữ liệu đặc biệt và có thể sử dụng chúng trong các tham số của chương trình con khác. Sau đây là ví dụ minh họa điều đó.

Ví dụ 20. Chương trình:

```
# ham bac nhat
def bac_nhat(x):
    return 2*x
# ham bac hai
def bac_hai(x):
    return x*x
# tham so la chuong trinh con
def ham(f, x):
    return f(x)
# goi chuong trinh con
print("ham bac nhat: ", ham(bac_nhat, 10))
print("ham bac hai: ", ham(bac_hai, 10))
```

☞ Kết quả:

```
ham bac nhat:  20
ham bac hai:  100
```

9. Sử dụng các hàm trong sắp xếp

Trong chủ đề 10 và chủ đề 11, đã đề cập đến sắp xếp các phần tử trong một danh sách. Trên thực tế, việc sắp xếp các phần tử có thể dựa vào nhiều tiêu chí khác nhau. Python cho phép sử dụng một hàm để tính các tiêu chí đó và xem như là khóa (key) đồng thời đưa hàm này vào lệnh **`sort()`** để sắp xếp.

Ví dụ 21. Cho dãy số nguyên dương $a_0, a_1, .., a_{n-1}$. Hãy sắp xếp dãy theo tiêu chí chữ số tận cùng tăng dần.

Chương trình:

```
def tc(x):
    return x%10
ds = [10, 13, 22, 32]
ds.sort(key = tc)
print(ds)
```

☞ Kết quả:

```
[10, 22, 32, 13]
```

❖ **Chú ý**: Trong ví dụ trên, hàm `tc(x)` trả về chữ số tận cùng của x. Khi thực hiện câu lệnh `ds.sort(key = tc)`, chương trình sẽ lấy từng phần tử trong danh sách và sử dụng hàm `tc()` để tính chữ số tận cùng và sử dụng giá trị này như là một khóa (*key*) để so sánh giữa các phần tử. Ngoài ra, ta cũng có thể sử dụng trực tiếp hàm `lambda` như sau:

```
ds.sort(key = lambda x: x%10)
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. *Viết chương trình con tìm giá trị lớn nhất của 2 số, của 3 số, của n số.*

Bài tập 2. *Viết chương trình con tính số Fibonacci thứ n (n là tham số) được định nghĩa:*

$F_1 = F_2 = 1, F_n = F_{n-1} + F_{n-2}$ *với* $n \geq 3$.

Bài tập 3. *Viết chương trình con in ra tam giác Pascal gồm* $n + 1$ *dòng.*

Ví dụ n = 3, ta có tam giác Pascal:

```
1
1 1
1 2 1
1 3 3 1
```

Bài tập 4. *Viết chương trình con tham số vào là một dãy số và trả về kết quả là một dãy số đại diện (mỗi phần tử chỉ xuất hiện một lần).*

Ví dụ: Cho danh sách [1, 3, 2, 1, 4, 4, 2], *danh sách kết quả* [1, 3, 2, 4].

Bài tập 5. Tham số là các chương trình con

Viết các chương trình con:

sum_square(*<biến kiểu danh sách>*)

Trả về tổng bình phương các phần tử trong <biến kiểu danh sách>.

sum_cube(*<biến kiểu danh sách>*)

Trả về tổng lập phương các phần tử trong <biến kiểu danh sách>.

sum_function(*<hàm>*, *<biến kiểu danh sách>*)

Trả về 2 lần kết quả của <hàm> ứng với tham số là <biến kiểu danh sách>.

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Giá trị lớn nhất:

- Giá trị lớn nhất của hai số

```
def max_two(x, y):
    if x > y: return x
    return y
```

- Giá trị lớn nhất của 3 số

```
def max_three(x, y, z):
    if x >= y and x >= z: return x
    if y >= x and y >= z: return y
    return z
```

- Giá trị lớn nhất của một danh sách

```
def max_list(x,*L):
    kq = x
    for y in L:
        if y > kq: kq = y
    return kq
```

Bài tập 2. Số Fibonacci thứ *n*

```
def  Fibo(n):
    if n <= 2: return 1
    a = b = 1
    i = 3
    while i <= n:
        c = a + b
        b = a
        a = c
        i += 1
    return c
```

Một lời giải đẹp:

```
def Fibo(n):
    a = b = 1
    i = 3
    while  i <= n:
        a, b = b, a + b
        i += 1
    return b
```

Bài tập 3. Tam giác Pascal

```
def  printList(L):
    for x in L:
        print(x, end = ' ')
    print()
#--
def  Pascal(n):
    list1 = [1]
    d = 0
```

```
    while  d <= n:
        printList(list1)
        list2 = []+list1                    #(1)
        for i in range(len(list1)-1):
            list1[i+1] = list2[i+1] + list2[i]
        list1 = list1+[1]                     #(2)
        d += 1
```

Bài tập 4: Danh sách đại diện

```
def Danh_Sach_Dai_Dien(List):
    Listkq = []
    for x in List:
        if x not in Listkq:
            Listkq.append(x)
    return Listkq
#Het chuong trinh con
List = [1, 2, 3, 2, 1, 2, 3, 6]
List1 = Danh_Sach_Dai_Dien(List)
print(List1)
```

Bài tập 5: Tham số là chương trình con

```
def sum_square(List):
    s = 0
    for x in List:
        s += x*x
    return s
#------
def sum_cube(List):
    s = 0
    for x in List:
```

```
        s+= x*x*x
    return s
#-------
def sum_function(f,List):
    return(f(List))
#-------
print(sum_square([1, 2, 3]))
print(sum_cube([1, 2, 3]))
print(sum_function(sum_square,[1, 2, 3]))
print(sum_function(sum_cube,[1, 2, 3]))
```

☛ Kết quả:

```
14
36
14
36
```

CHỦ ĐỀ 17. CHƯƠNG TRÌNH CON ĐỆ QUY, ĐỆ QUY QUAY LUI VÀ ĐỆ QUY CÓ NHỚ

A. KIẾN THỨC CƠ BẢN

1. Định nghĩa theo đệ quy

Một khái niệm X được gọi là định nghĩa theo đệ quy nếu trong định nghĩa X có sử dụng khái niệm X.

Ví dụ 1. Định nghĩa về số tự nhiên:

+ 0 là số tự nhiên.

+ n là số tự nhiên nếu n – 1 là số tự nhiên (*).

Như vậy, khái niệm số tự nhiên ở vị trí (*) được sử dụng lại. Với cách định nghĩa theo đệ quy này, ta dễ dàng suy ra tập các số tự nhiên là: 1, 2, 3, ...

Ví dụ 2. Định nghĩa về giai thừa:

Với một số tự nhiên n, n giai thừa được kí hiệu $n!$ và được xác định:

+ $0! = 1$

+ $n! = n * (n-1)!$

Như vậy khái niệm $n!$ được định nghĩa thông qua khái niệm $(n-1)!$. Với cách định nghĩa theo đệ quy này, ta tính được $n! = 1 \times 2 \times 3 \times..\times n$.

Ví dụ 3. Cho tập A được xác định:

+ $1 \in A$;

+ $x \in A$ nếu $x - 2 \in A$;

Với cách định nghĩa theo đệ quy để xác định tập A, ta nhận thấy tập A gồm các số tự nhiên lẻ: 1, 3, 5, 7, ..

2. Chương trình con đệ quy

Khi một khái niệm hay một đối tượng được xác định theo đệ quy, để xác định chúng, ta có thể sử dụng chương trình con đệ quy.

Chương trình con đệ quy là chương trình con mà trong thân của nó có lời gọi tới chính chương trình con đó.

Ví dụ 4. Chương trình:

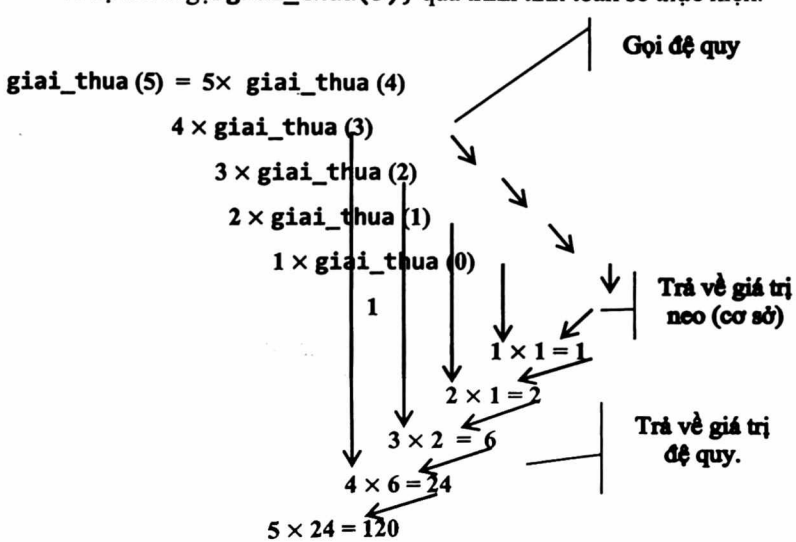
```
def giai_thua(n):
    if n== 0: return 1          #phan neo
    else:
        x = giai_thua(n-1)  #phan de quy
        return n*x
#Het chuong trinh con
print("5 giai thua bang: ",giai_thua(5))
print("20 giai thua bang: ",giai_thua(20))
```

☞ Kết quả:

```
5 giai thua bang:  120
20 giai thua bang:  2432902008176640000
```

Quá trình tính toán trong đệ quy:

Ví dụ khi ta gọi **giai_thua(5)**, quá trình tính toán sẽ thực hiện:



3. Cấu trúc chương trình con đệ quy

Một chương trình con đệ quy bao giờ cũng có hai phần: Phần cố định (*phần neo*) và phần đệ quy.

• Phần cố định dùng để xác định giá trị ban đầu của khái niệm đệ quy. Ở ví dụ về định nghĩa giai thừa. Giá trị ban đầu của khái niệm đó là 0! = 1. Trong chương trình con ở trên, phần cố định chính là câu lệnh: `if n == 0: return 1 #phan neo`

• Phần đệ quy dùng để gọi tới chính chương trình con đó. Ở phần này, lời gọi tới chương trình con đó với tham số nhỏ hơn. Việc gọi đệ quy này sẽ dừng lại khi gặp phần cố định. Trong chương trình con ở trên, phần đệ quy chính là câu lệnh:

```
x = giai_thua(n-1)        #phan de quy
```

4. Phía sau lời gọi đệ quy (tail recursion)

Ta hãy xem kết quả của ví dụ sau để hiểu sự thực hiện của các câu lệnh sau lời gọi đệ quy.

Ví dụ 5. Chương trình:

```
def chuyendoi(n):
    if n > 0:
        chuyendoi(n//3)      #Loi goi de quy
        print(n%3,end = '') #Cau lenh phia sau de quy
                             #Het chuong trinh con
chuyendoi(15)
```

☞ Kết quả:

120

Đây là chương trình con chuyển đổi số nguyên dương n ở hệ thập phân sang hệ tam phân (hệ cơ số 3). Thuật toán thực hiện bằng cách chia liên tiếp cho 3 cho đến khi thương bằng 0. Các số dư lần lượt nhận được khi viết theo thứ tự ngược lại thì ta được kết quả.

Với $n = 15$, quá trình được thực hiện:

15 chia 3: thương 5, dư 0.

5 chia 3: thương 1, dư 2.

1 chia 3: thương 0, dư 1 (thuật toán kết thúc).

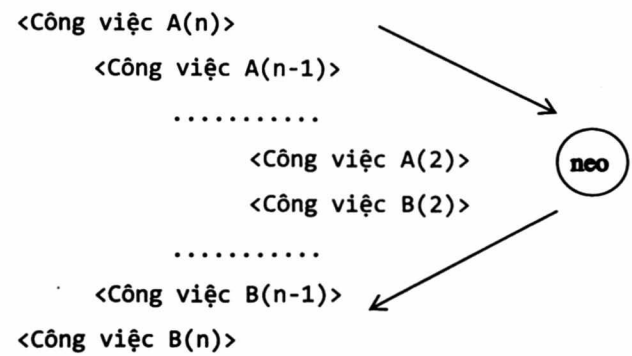
Các số dư được viết theo thứ tự ngược lại: 120.

Như vậy, ta thấy, câu lệnh đệ quy sẽ được gọi cho đến khi gặp phần neo (không gọi tiếp) thì các câu lệnh ở sau câu lệnh gọi đệ quy mới được thực hiện. Hơn nữa thứ tự thực hiện được diễn ra như sau:

Giả sử chương trình con có cấu trúc:

```
def subprogram(n):
    if  n == 0: return
    <Công việc A(n)>        #Truoc de quy
    subprogram(n-1)           #Goi de quy
    <Công việc B(n)>        #Sau de quy
```

Thứ tự thực hiện:



5. Đệ quy quay lui – Backtracking

Đệ quy quay lui là sự kết hợp giữa đệ quy và sự quay lui. Sự kết hợp đó tạo nên một phương pháp giải các bài toán rất hiệu quả. Sau đây là một số ví dụ cụ thể sử dụng đệ quy quay lui.

Ví dụ 6. Đưa ra tất cả các dãy nhị phân độ dài n. Chẳng hạn với $n = 3$, ta có các dãy nhị phân độ dài 3: '000', '001', '010', '011', '100', '101', '110', '111'.

Ta biểu diễn dãy nhị phân độ dài n như một danh sách gồm n thành phần $x[0], x[1], .., x[n-1]$ với $x[i] = 0$ hoặc 1. Việc xây dựng các giá trị của x[i] được thực hiện rất hiệu quả bằng đệ quy quay lui.

Chương trình:

```
n = 3
x = [0 for i in range(n)]
def  Nhiphan(i):
   for t in range(2):
      x[i] = t
      if i == n-1:
         print(x)
      else:
         Nhiphan(i+1)
Nhiphan(0)
```

☞ Kết quả:

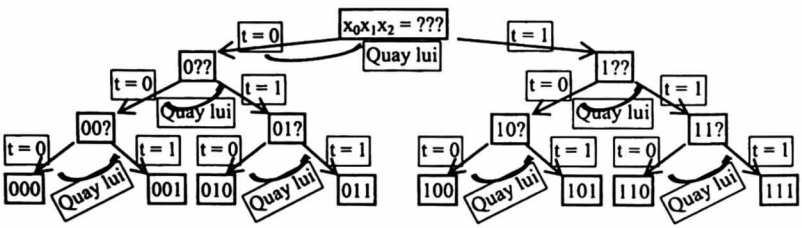
```
[0, 0, 0]
[0, 0, 1]
[0, 1, 0]
[0, 1, 1]
[1, 0, 0]
[1, 0, 1]
[1, 1, 0]
[1, 1, 1]
```

Sự quay lui và đệ quy được thực hiện trong các câu lệnh:

```
for t in range(2):
      x[i] = t
      if i == n-1:
         print(x)
      else:
         Nhiphan(i+1)
```

Trong vòng **for**, giá trị của t sẽ lần lượt nhận là 0 và 1. Khi t nhận giá trị 0, nếu $i < n-1$, chương trình sẽ gọi đệ quy `Nhiphan(i+1)`. Và cứ tiếp tục như vậy cho đến khi $i == n-1$, chương trình sẽ quay lui lại để cho t nhận giá trị bằng 1.

Cây đệ quy quay lui



Sơ đồ chung của chương trình con đệ quy quay lui

Giả sử bài toán cần tính n thành phần $x[0], x[1], .., x[n-1]$. Giá trị của các thành phần $x[i]$ là tập các giá trị $D[i]$. Sơ đồ chung của chương trình con đệ quy quay lui như sau:

```
def Try(i):
    for a in D[i]:
        if <x[i] nhận được giá trị a ?>:
            x[i] = a
            if(i == n-1):
                <Hoàn thành một bộ nghiệm>
            else:
                Try(i+1)
#Bat dau xay dung thanh phan dau tien
Try(0)
```

6. Đệ quy có nhớ

Ví dụ 7. Ta xét bài toán tìm số hạng Fibonacci thứ n:

$$F_1 = F_2 = 1, F_n = F_{n-1} + F_{n-2}.$$

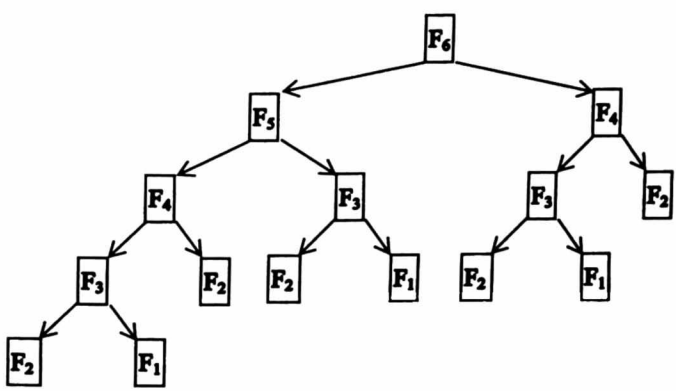
Chương trình đệ quy cho phép tính F_n như sau:

```
def Fibo(n):
    if n <= 2:
        return 1
    else:
        return Fibo(n-1) + Fibo(n-2)
n = 10
print(Fibo(n))
```

☞ Kết quả:

```
55
```

Khi ta cho giá trị n = 100, thì thấy chương trình chạy rất lâu. Điều này được lý giải là do sự tính lại của quá trình đệ quy bị diễn ra rất nhiều. Chẳng hạn với n = 6, cây đệ quy tính F_6 sẽ là:



Để khắc phục điều này, ta có thể sử dụng phương pháp ghi nhớ các lời giải đã được tính, khi cần lấy kết quả sẽ không phải tính lại nữa. Đó chính là nội dung của đệ quy có nhớ.

Trong bài này, ta có thể sử dụng một danh sách $F[1], F[2], .., F[n]$ để lưu các giá trị của các số hạng Fibonacci. Nếu $F[i] = 0$ có nghĩa là F_i chưa được tính.

Chương trình đệ quy có nhớ để tính số Fibonacci:

```
n = 100
F = [0 for i in range(n+1)]
def  Fibo(n):
   if n <= 2:
      F[n] = 1
      return 1
   #Neu F[n-1] chua tinh thi goi de quy Fibo(n-1)
de tinh F[n-1]
   if F[n-1] == 0:
      F[n-1] = Fibo(n-1)
   #Neu F[n-1] chua tinh thi goi de quy Fibo(n-1)
de tinh F[n-1]
   if F[n-2] == 0:
      F[n-2] = Fibo(n-2)
   return F[n-1] + F[n-2]
x = Fibo(n)
print(x)
```

☛ Kết quả:

```
354224848179261915075
```

❖ Sơ đồ chung của chương trình con đệ quy có nhớ

Giả sử ta cần giải bài toán $F(n)$. Trong quá trình đệ quy tìm lời giải $F(n)$, ta cần lời giải các bài toán $F(n_1)$, $F(n_2)$, .. , $F(n_k)$. Trong các bài toán $F(n_1)$, $F(n_2)$, .. , $F(n_k)$, nếu bài toán nào chưa được giải thì sẽ gọi đệ quy để tìm lời giải. Điều quan trọng ở đây là khi giải được một bài toán nào đó thì phải lưu lại lời giải đó để lần sau nếu cần sử dụng thì không phải giải lại mà lấy trực tiếp lời giải đã lưu trữ trước đó. Sau đây là sơ đồ chung của giải bài toán theo đệ quy có nhớ.

```
<Đánh dấu tất cả các bài toán đều chưa được giải>
def SolveF(n):
```

```
    for i in <Tập các bài toán cần sử dụng lời giải>:
        if <F[i] chưa giải>:
            F[i] = SolveF(i)  #Gọi đệ quy để giải F[i]
        else:
            <Sử dụng lời giải F[i] để giải F[n]>
F[n] = SolveF(n)
```

7. Ưu điểm và nhược điểm của đệ quy

- *Ưu điểm*

Khi sử dụng đệ quy để giải các bài toán, ta thấy nhiều trường hợp tỏ ra rất hiệu quả, rõ ràng và dễ viết chương trình. Sử dụng đệ quy giúp ta trừu tượng hóa được nhiều vấn đề mà nếu không sử dụng chúng thì rất khó cài đặt và diễn đạt, chẳng hạn như các bài toán về đệ quy quay lui.

- *Nhược điểm*

Bên cạnh những ưu điểm, đệ quy có những nhược điểm về bộ nhớ stack của máy tính. Khi thực hiện gọi đệ quy, các biến được khai báo trong chương trình con được nhân lên (được cấp bộ nhớ) và điều này có thể dẫn đến thiếu bộ nhớ khi thực hiện gọi đệ quy nhiều lần.

Ngoài ra, khi thực hiện tính toán bằng đệ quy sẽ chậm hơn nếu chúng ta không sử dụng đệ quy.

Ví dụ 8. Hãy chạy hai chương trình cùng tính giá trị F_{10000}.

Chương trình 1

```
n = 10000
F = [0 for i in range(n+1)]
def  Fibo(n):
    if n <= 2:
        F[n] = 1
        return 1
    if F[n-1] == 0:
        F[n-1] = Fibo(n-1)
```

```
        if F[n-2] == 0:
            F[n-2] = Fibo(n-2)
        return F[n-1] + F[n-2]
    print(Fibo(n))
```

Chương trình 2

```
n = 10000
F = [0 for i in range(n+1)]
F[1] = F[2] = 1
for i in range(3, n+1):
    F[i] = F[i-1] + F[i-2]
print(F[n])
```

Ta nhận thấy, chương trình 2 chạy rất tốt, còn chương trình 1 r
lỗi: `RecursionError: maximum recursion depth exceeded.`

8. Tăng thêm độ sâu đệ quy

Ta xét chương trình tính tổng 1 + 2 + .. + n bằng đệ quy như sa

```
def sum_recur(n):
    if n == 1:
        return 1
    else:
        return n + sum_recur(n-1)
```

Ta nhận thấy, khi gọi `sum_recur(998)`, chương trình chạy chc
quả 498501, nhưng khi gọi `sum_recur(999)`, `sum_recur(1000)`
chương trình sinh lỗi **maximum recursion depth exceeded.** I
này được lý giải là Python mặc định không cho quá 1000 lần gọi đệ
trong một chương trình con đệ quy. Để biết được số lần gọi đệ quy tố
ta sử dụng lệnh **`sys.getrecursionlimit()`**.

Ví dụ 9. Chương trình:

```
import sys
a = sys.getrecursionlimit()
print("So lan toi da goi de quy: ", a)
```

☞ Kết quả:

```
So lan toi da goi de quy:  1000
```

Thay đổi số lần tối đa gọi đệ quy

Ta có thể thay đổi số lần tối đa gọi đệ quy bằng câu lệnh `sys.setrecursionlimit([số lần tối đa])`.

Ví dụ 10. Chương trình:

```
import sys
sys.setrecursionlimit(10000)
a = sys.getrecursionlimit()
print("So lan toi da goi de quy: ", a)
```

☞ Kết quả:

```
So lan toi da goi de quy:  10000
```

Bây giờ ta xét lại chương trình tính tổng 1 + 2 + .. + n bằng đệ quy:

```
import sys
sys.setrecursionlimit(10000)
def sum_recur(n):
    if n == 1:
        return 1
    else:
        return n + sum_recur(n-1)
n = 5000
print("Tong  1+2+..+  ", n, "  =  ",
sum_recur(n))
```

☞ Kết quả:

```
Tong 1+2+..+  5000  =  12502500
```

Tuy nhiên, khi cho giá trị n = 9000, chương trình sẽ sinh lỗi không đủ bộ nhớ stack, `MemoryError: Stack overflow`.

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tính tổng các số hạng trong một danh sách bằng đệ quy

Cho danh sách $L[0], L[1], .., L[n-1]$. Hãy tính tổng các phần tử của danh sách này.

Bài tập 2. Ước chung lớn nhất của hai số nguyên dương bằng đệ quy

Cho hai số nguyên dương a và b. Tìm ước chung lớn nhất của a và b.

Bài tập 3. Tính lũy thừa a^n bằng phương pháp đệ quy

Cho hai số nguyên dương a và n. Tính a^n.

Bài tập 4. Bài toán liệt kê hoán vị

Cho số nguyên dương n. Hãy liệt kê tất cả các hoán vị của n số tự nhiên $1, 2, .., n$. Ví dụ $n = 3$. Ta có 6 hoán vị: $[1,2,3]$, $[1,3,2]$, $[2,1,3]$, $[2,3,1]$, $[3,1,2]$, $[3,2,1]$.

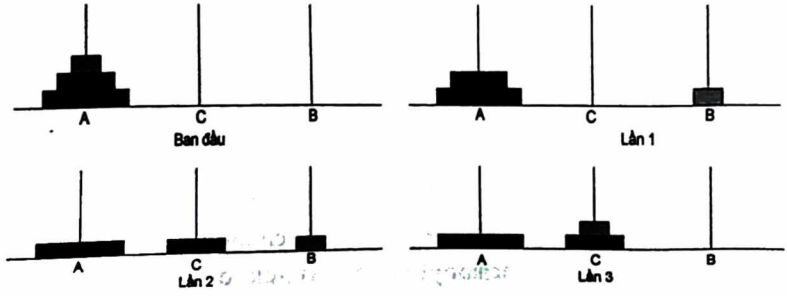
Bài tập 5. Bài toán tháp Hà Nội

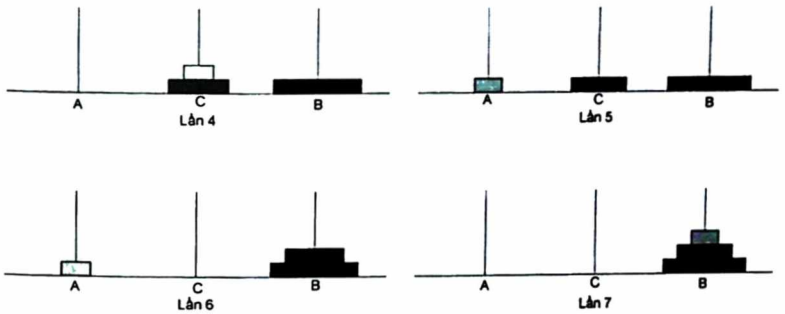
Có 3 chiếc cột được đánh số là A, B, C. Ban đầu, cột A có n chiếc đĩa (có lỗ tròn ở giữa), n đĩa được đánh số thứ tự từ 1 đến n (từ trên xuống dưới), đĩa ở trên luôn nhỏ hơn đĩa ở dưới.

Yêu cầu: Hãy chuyển n đĩa ở cột A sang cột B.

Quy tắc chuyển: Có sử dụng cột C làm cột trung gian. Trong quá trình chuyển, không đặt đĩa có kích thước lớn lên đĩa có kích thước nhỏ.

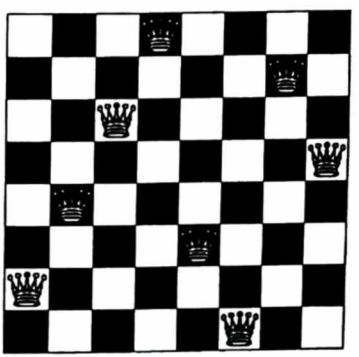
Ví dụ, với $n = 3$, yêu cầu: Chuyển 3 đĩa từ cột A sang cột B, ta cần thực hiện 7 lần chuyển đĩa như sau:





Bài tập 6. Bài toán xếp hậu

Xét bàn cờ vua gồm 8 hàng, 8 cột. Hãy xếp 8 quân hậu lên bàn cờ sao cho không có hai quân hậu nào khống chế nhau. Dưới đây là một cách xếp hậu.



C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tính tổng các số hạng trong một danh sách bằng đệ quy

Ý tưởng thuật toán

Ta gọi S_i là tổng $L[0] + L[1] + .. + L[i]$. Như vậy tổng cần tính bằng S_{n-1}.

Nhận thấy nếu $i = 0$ thì $S_0 = L[0]$, ngược lại ($i > 0$) thì $S_i = S_{i-1} + L[i]$. Phân tích này cho chúng ta có thể sử dụng đệ quy để tính tổng $L[0] + L[1] + .. + L[n-1]$ như sau:

Chương trình

```
def Tong_day(n):
    if n == 1:
        return L[0]
    else:
        return L[n-1] + Tong_day(n-1)
n = int(input("Nhap so phan tu: "))
print("Nhap ",n," phan tu: ")
L = [int(input()) for i in range(n)]
print("Tong cac so hang nhap vao:
",Tong_day(n))
```

Bài tập 2. Ước chung lớn nhất của hai số nguyên dương bằng đệ quy

Ý tưởng thuật toán

Ta nhận thấy: $UCLN(a,b) = UCLN(b,r)$ với r là số dư của a chia cho b ($r = a\%b$). Nếu $r = 0$ thì $UCLN(b,r) = b$ và kết thúc, ngược lại ta thay $a = b, b = r$ và tiếp tục thực hiện như trên.

Chương trình

```
def UCLN(a, b):
    if b == 0:
        return a
    else:
        return UCLN(b, a%b)
print("Nhap hai so a va b")
a = int(input())
b = int(input())
print("Uoc chung lon nhat (a, b) = ",UCLN(a,b))
```

Bài tập 3. Tính lũy thừa a^n bằng phương pháp đệ quy

Ý tưởng thuật toán

Ta gọi T_i là a^i. Nhận thấy $T_1 = a$, $T_i = a \times T_{i-1}$ và kết quả cần tính là T_n.

Chương trình

```
def LuyThua(a, n):
    if n == 1:
        return a
    else:
        return a*LuyThua(a, n-1)
print("Nhap hai so nguyen duong a va n")
a = int(input())
n = int(input())
print("a^n = ", LuyThua(a, n))
```

Cách tiếp cận bằng phương pháp chia để trị (*divide and conquer*)

Nhận thấy, số lần gọi đệ quy trong chương trình trên là n. Khi n lớn sẽ dẫn đến hai vấn đề:

- Số lần tính toán sẽ lâu.

- Chương trình gặp lỗi: `RecursionError: maximum recursion depth exceeded in comparison.`

Để khắc phục các vấn đề này, ta sử dụng phương pháp chia để trị như sau:

- Với n = 1, thì $a^n = a^1 = a$.

- Với n là một số chẵn:

Thì có: $a^n = a^m \times a^m$ với m = n//2.

- Do vậy ta chỉ việc tính $u = a^m$, sau đó thực hiện $u \times u = a^n$.

Với n là một số lẻ:

Thì có: $a^n = a^m \times a^m \times a$ với m = n//2.

Do vậy ta chỉ việc tính $u = a^m$, sau đó thực hiện $u \times u \times a = a^n$.

Ví dụ: Tính 2^{10}:

Ta có: $2^{10} = (2^5) \times (2^5)$, tính 2^5?

$2^5 = (2^2) \times (2^2) \times 2$, tính 2^2?

$2^2 = (2^1) \times (2^1)$, $2^1 = 2$.

Như vậy, việc tính 2^{10} được thực hiện:

$2^1 = 2$

$2^2 = (2^1) \times (2^1) = 2 \times 2 = 4$

$2^5 = (2^2) \times (2^2) \times 2 = 4 \times 4 \times 2 = 32$

$2^{10} = (2^5) \times (2^5) = 16 \times 16 = 1024$.

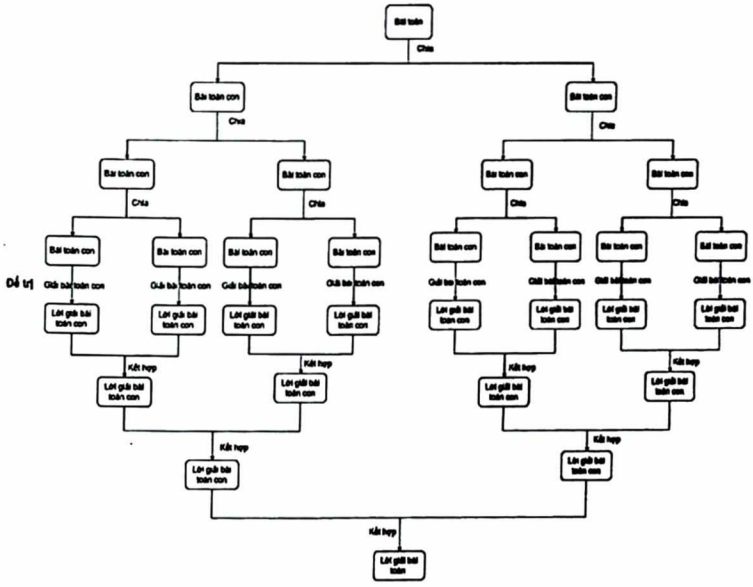
Chương trình

```
def LuyThua(a, n):
    if n == 1:
        return a
    else:
        x = LuyThua(a, n//2)
        if n%2 == 0:
            return x*x
        else:
            return x*x*a
print("Nhap hai so nguyen duong a va n")
a = int(input())
n = int(input())
print("a^n = ", LuyThua(a, n))
```

❖ ***Nhận xét***

Số phép tính (số lần gọi đệ quy) trong chương trình trên khoảng là $\log_2 n$. *Chẳng hạn* $n = 10^9$, *số lần tính toán khoảng 30 lần, ít hơn rất nhiều so với* 10^9.

❖ **Sơ đồ chung trong cách tiếp cận giải bài toán bằng phương pháp chia để trị**



Bài tập 4. Bài toán liệt kê hoán vị

Cho số nguyên dương n. Hãy liệt kê tất cả các hoán vị của n số tự nhiên.

Ý tưởng thuật toán

Bài toán liệt kê hoán vị được thực hiện rất hiệu quả khi sử dụng đệ quy quay lui.

Ta biểu diễn một hoán vị là một danh sách gồm n phần tử $L[0], L[1], .., L[n-1]$. Để được một hoán vị, chương trình sẽ tiến hành tìm các giá trị cho $L[0], L[1], .., L[n-1]$.

Nhận thấy $L[i]$ ($i = 0, 1, .., n-1$) đều có thể nhận giá trị $1, 2, .., n$. Tuy nhiên khi $L[i] = a$ thì $L[j] \neq a$ với $i \neq j$.

Với mỗi giá trị $a = 1, 2, .., n$ sẽ có hai trạng thái: đã được sử dụng và chưa được sử dụng. Nếu a được gán cho L[i] nào đó, nghĩa là a đã được sử dụng và không được sử dụng gán cho $L[j]$ với $i \neq j$.

Do vậy để tính giá trị cho $L[i]$, ta lần lượt duyệt qua tất cả các giá trị 1, 2, .., n – 1. Nếu giá trị a chưa được sử dụng thì $L[i] = a$ và đánh dấu giá trị a đã được sử dụng.

Khi tính được $L[i]$, ta tiếp tục gọi đệ quy để tính $L[i+1]$. Quá trình đệ quy thực hiện cho đến khi tính được $L[n-1]$, thì có một hoán vị.

Sự quay lui thể hiện ở chỗ: $L[i]$ không chỉ nhận một giá trị là a mà có thể nhận nhiều giá trị khác. Vì vậy, sự quay lui để tính hết tất cả các giá trị mà $L[i]$ có thể nhận.

Chú ý rằng khi $L[i] = b, (b \neq a)$ thì giá trị a có thể được gán cho các $L[i+1], .., L[n-1]$.

Chương trình

```
n = int(input("Nhap n = "))
#used[i] == 0 <-> i chua su dung
#used[i] == 1 <-> i da su dung
used = [0 for i in range(n+1)]
#Danh sach luu hoan vi
L    = [0 for i in range(n)]
def HoanVi(i):
   for a in range(1, n+1):
      if used[a] == 0: #a chua su dung
         L[i] = a       #Tinh L[i] = a
         used[a] = 1    #Xac nhan a da su dung
         if i == n-1:      #Da duoc mot hoan vi
            print(L)
         else:
            HoanVi(i+1)    #Goi de quy
         used[a] = 0    #Tra trang thai su dung
HoanVi(0)
```

❖ **Sơ đồ chung của đệ quy quay lui có thay đổi trạng thái**

Giả sử bài toán cần tính n thành phần $x[0], x[1], .., x[n-1]$. Giá trị của các thành phần $x[i]$ là tập các giá trị $D[i]$, mỗi giá trị trong tập chỉ được sử dụng cho một thành phần. Vì vậy, ta cần đánh dấu để không sử dụng lặp lại. Sơ đồ chung của chương trình con đệ quy quay lui có thay đổi trạng thái như sau:

```
<Đánh dấu tất cả các giá trị a chưa được sử dụng>
def Try_TrangThai(i):
    for a in D[i]:
        if <x[i] nhận được giá trị a ?>:
            x[i] = a
            <Đánh dấu trạng thái đã sử dụng giá trị a>
            if(i == n-1):
                <Hoàn thành một bộ nghiệm>
            else:
                Try_TrangThai(i+1)
            <Trả lại trạng thái chưa sử dụng giá trị a>
#Bat dau xay dung thanh phan dau tien
Try_TrangThai(0)
```

Bài tập 5. Bài toán tháp Hà Nội

Ý tưởng thuật toán

Ta nhận thấy, với $n = 2$, lời giải:

Chuyển đĩa 1 từ cột A sang cột C.

Chuyển đĩa 2 từ cột A sang cột B.

Chuyển đĩa 1 từ cột C sang cột B và ta có được lời giải.

Với $n = 3$, ta thực hiện:

- Chuyển đĩa 1 và đĩa 2 sang cột C (sử dụng cột B như là cột trung gian).
- Chuyển đĩa 3 từ cột A sang cột B.

- Chuyển đĩa 1, 2 từ cột C sang cột B (sử dụng cột A như là cột trung gian).

Với $n \geq 4$, ta thực hiện:

- Chuyển các đĩa 1, 2, .., $n - 1$ từ cột A sang cột C (sử dụng cột B như là cột trung gian).
- Chuyển đĩa n từ cột A sang cột B.
- Chuyển các đĩa 1, 2, .., $n - 1$ từ cột C sang cột B (sử dụng cột A như là cột trung gian).

Như vậy, lời giải được xây dựng trên chiến lược chia để trị và sử dụng đệ quy để thực hiện.

Chương trình

```
def Chuyen_dia(n, A , B, C):
    if n == 1:
        print("Chuyen dia: ",n," tu ", A ," --> ", B)
    else:
        Chuyen_dia(n - 1, A, C, B)
        print("Chuyen dia: ",n," tu ", A ," --> ", B)
        Chuyen_dia(n - 1, C, B, A)
n = int(input('Nhap so dia can chuyen: '))
Chuyen_dia(n,"A","B","C")
```

Bài tập 6. Bài toán xếp hậu

Ý tưởng thuật toán

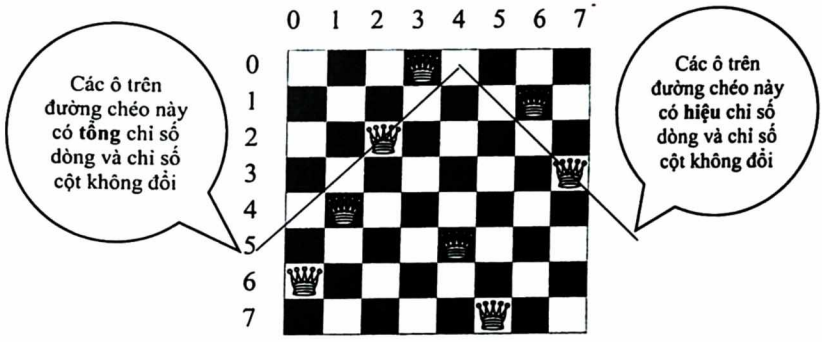
Ta đánh chỉ số dòng của bàn cờ là: 0, 1, 2, 3, 4, 5, 6, 7; và chỉ số cột của bàn cờ là: 0, 1, 2, 3, 4, 5, 6, 7.

Nhận thấy, mỗi dòng chỉ đặt được duy nhất một quân hậu, do vậy, ta có thể biểu diễn một cách xếp hậu là một danh sách gồm 8 phần tử x[0], x[1], .. , x[7] với ý nghĩa 8 quân hậu được đặt tại 8 ô: (0, x[0]); (1, x[1]); (2, x[2]); (3, x[3]); (4, x[4]); (5, x[5]); (6, x[6]); (7, x[7]).

Ta sẽ tìm các giá trị x[0], x[1], .. , x[7] bằng phương pháp đệ quy quay lui có ghi nhận trạng thái.

Với mỗi x[i], i = 0, 1, .., 7, giá trị x[i] $\in \{0, 1, 2, 3, 4, 5, 6, 7\}$, tuy nhiên x[i] $\neq$ x[j] với i $\neq$ j, nghĩa là trên cột i chỉ có duy nhất một quân hậu.

Để hai quân hậu không khống chế nhau thì trên mỗi đường chéo của bàn cờ có nhiều nhất một quân hậu.



Do vậy, ta phải sử dụng 3 danh sách trạng thái:

$Cot[], CheoTong[], CheoHieu[]$ với ý nghĩa:

$Cot[i] = True$ (i = 0, 1, .. , 7) mô tả cột i đã đặt hậu.

$CheoTong[t] = True$ (t = 0, 1, 2, .. , 14) mô tả đường chéo chứa ô (i, j) với $i + j = t$ đã đặt hậu.

$CheoHieu[t] = True$ (t = -7, -6, .., 0, 1, 2, .. , 7) mô tả đường chéo chứa ô (i, j) với $i - j = t$ đã đặt hậu. Ở đây, do các chỉ số trong danh sách không nhận giá trị âm nên ta phải cộng các chỉ số với 7 để nhận được các chỉ số 0 âm.

Ban đầu các danh sách $Cot[], CheoTong[], CheoHieu[]$ có các phần tử đều nhận giá trị $False$.

Chương trình

```
x         = [0 for i in range(8)]
Cot       = [False for i in range(8)]
CheoTong = [False for i in range(15)]
CheoHieu = [False for i in range(15)]
def InCachXep():
   for i in range(8):
      print("(",i," ", x[i],")",end = " ")
   print()
def XepHau(i):
   for j in range(8):
      if Cot[j] == False and CheoTong[i+j] == False
and     CheoHieu[i-j+7] == False:
         x[i] = j
         Cot[j] = CheoTong[i+j] = CheoHieu[i-j+7] = True
         if i == 7:
            InCachXep()
         else:
            XepHau(i+1)
         Cot[j] = CheoTong[i+j] = CheoHieu[i-j+7] = False
XepHau(0)
```

CHỦ ĐỀ 18. KIỂU DỮ LIỆU LỚP - CLASS

Cũng giống như nhiều ngôn ngữ lập trình khác, ngoài các kiểu dữ liệu đã cung cấp bởi ngôn ngữ lập trình, các ngôn ngữ lập trình đều cho phép người lập trình tổ chức dữ liệu theo cách riêng của mình để thuận lợi trong việc lưu trữ cũng như xử lý cho từng bài toán cụ thể. Python cũng vậy, kiểu dữ liệu `class` cho phép người lập trình tổ chức dữ liệu theo cách của mình, phù hợp trong lưu trữ và xử lý dữ liệu. Ngoài ra, kiểu `class` cho phép lập trình hướng đối tượng (Object Oriented Programming - OOP).

A. Kiến thức cơ bản

1. Định nghĩa lớp và tạo một đối tượng kiểu lớp

Mỗi lớp được thiết kế như là một thực thể thống nhất, chúng gồm có hai thành phần chính:

- *Thuộc tính:* là các biến dữ liệu, dùng để lưu trữ các dữ liệu, thông tin để mô tả thực thể đó.

- *Phương thức:* là các chương trình con, dùng để tính toán những kết quả cần thiết, dựa trên những dữ liệu được lưu trữ trong các thuộc tính.

Ví dụ, ta có thể xây dựng một kiểu dữ liệu lớp "circle" để lưu các thông tin về một đường tròn (thuộc tính), và các chương trình con (phương thức) cho phép tính chu vi, diện tích của đường tròn đó.

- Định nghĩa một lớp

```
class    <Tên lớp>:
    [Khối lệnh]
```

- Tạo một đối tượng kiểu lớp

```
<Tên đối tượng> = <Tên lớp>([Các tham số])
```

Ví dụ 1. Chương trình:

```
#Dinh nghia lop point
class point:
    x = 5
    y = 10
```

```
#Tao doi tuong p1 kieu point
p1 = point()
print("x = ",p1.x)
print("y = ",p1.y)
```

☞ Kết quả:

```
x =  5
y =  10
```

Ở ví dụ này, ta xây dựng lớp `point` có hai thuộc tính x và y.

Ví dụ 2. Chương trình:

```
class point():
    x = 5
    y = 10
    def infor(self):
        print("x = ",self. x)
        print("y = ",self. y)

p = point()
p.infor()
```

☞ Kết quả:

```
x =  5
y =  10
```

Ở đây, ta xây dựng một lớp `point` có hai thuộc tính x, y và m phương thức `infor()` dùng để in ra giá trị thuộc tính x, y.

2. Hàm tạo của một lớp

- Cách viết hàm tạo

```
class <tên lớp>:
    def __init__(self, [các tham số]):
        [Khối lệnh]
```

Ví dụ 3. Chương trình:

```
class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

p1 = point(10, 20)
p2 = point(15, 25)
print("p1.x = ", p1.x)
print("p1.y = ", p1.y)
print("p2.x = ", p2.x)
print("p2.y = ", p2.y)
```

☞ Kết quả:

```
p1.x =  10
p1.y =  20
p2.x =  15
p2.y =  25
```

Ví dụ 4. Chương trình:

```
class point():
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def infor(self):
        print("x = ",self. x)
        print("y = ",self. y)
p = point(10, 20)
p.infor()
```

☞ Kết quả:

```
x =  10
y =  20
```

🖮 Cơ chế hoạt động của hàm tạo

Mỗi khi một đối tượng lớp được tạo ra, hàm tạo **__init__()** sẽ tự động được thực hiện.

Ví dụ như đối tượng `p1 = point(10, 20)`, hàm tạo sẽ tự động thực hiện với tham số x = 10 và tham số y = 20.

🖮 Tham số **self**

Tham số **self** được sử dụng để tham chiếu đến tường đối tượng được tạo ra. Tên "**self**" cũng có thể thay thế bởi một tên khác nhưng ý nghĩa của tham số này không thay đổi.

Ví dụ 5. Chương trình:

```
class person:
   def __init__(self1, name, age):
      self1.name = name
      self1.age = age
   def infor(self2):
         print("Name: ",self2.name)
         print("Age: ",self2.age)
p1 = person("Phan Xuan Vong", 35)
p1.infor()
```

☞ Kết quả:

```
Name:  Phan Xuan Vong
Age:  35
```

❖ ***Nhận xét***

Tên **self** *đã được thay bởi tên* **self1** *và* **self2**. *Tuy nhiên, ta nên sử dụng tên* **self** *vì nó có tính gọi nhớ cao.*

3. Thay đổi giá trị của các thuộc tính

Các giá trị thuộc tính của một đối tượng có thể thay đổi trong khi thực hiện chương trình.

Ví dụ 6. Chương trình:

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def infor(self):
            print("Name: ",self.name)
            print("Age: ",self.age)
p1 = person("Phan Xuan Vong", 35)
p1.infor()
p1.age = 40
print("Thong tin sau khi cap nhat:")
p1.infor()
```

☞ Kết quả:

```
Name:  Phan Xuan Vong
Age:  35
Thong tin sau khi cap nhat:
Name:  Phan Xuan Vong
Age:  40
```

4. Xóa một đối tượng

Khi một đối tượng không cần cho việc lưu trữ thông tin, ta có thể xóa chúng để giải phóng bộ nhớ máy tính.

del <tên đối tượng>

Ví dụ 7. Chương trình:

```
class person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def infor(self):
            print("Name: ",self.name)
            print("Age: ",self.age)
p1 = person("Phan Xuan Vong", 35)
del p1
p1.infor()
```

☞ Kết quả:

Chương trình sẽ báo lỗi ở câu lệnh: `p1.infor()`.

```
NameError: name 'p1' is not defined
```

5. Sự kế thừa và bổ sung thêm phương thức, thuộc tính

Giả sử ta đã xây dựng một lớp A. Khi đó ta có thể sử dụng lớp A như là một thành phần để xây dựng nên lớp B. Việc sử dụng lớp A trong xây dựng lớp B được gọi là sự kế thừa từ lớp A, lớp A được gọi là lớp con của B, lớp B gọi là lớp cha của A.

Ta xét ví dụ, ở trên ta đã xây dựng lớp **person**, bây giờ ta xây dựng lớp **student**. Lớp **student** sẽ kế thừa các thuộc tính **name**, **age** từ lớp **person**, ta bổ sung thêm thuộc tính **graduation** để lưu thông tin về năm tốt nghiệp của sinh viên.

Ví dụ 8. Chương trình:

```
class person():
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

```
    def infor(self):
            print("Name: ",self.name)
            print("Age: ",self.age)
#Xay dung lop student
class student(person):
    def __init__(self, name, age, year):
        person.__init__(self, name, age)
        self.graduation = year
    def infor(self):
        person.infor(self)
        print("Graduation year: ",
self.graduation)
p1 = student("Phan Xuan Vong", 35, 2000)
p1.infor()
```

☞ Kết quả:

```
Name:  Phan Xuan Vong
Age:  35
Graduation year:  2000
```

6. Hàm super()

Ở ví dụ 8, khi sử dụng phương thức **infor()** của lớp **person**, ta phải viết *person.infor*(**self**), để không phải viết thêm tên lớp **person**, ta có thể sử dụng hàm **super()** như sau:

Ví dụ 9. Chương trình:

```
class person():
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def infor(self):
            print("Name: ",self.name)
            print("Age: ",self.age)
```

```
#Xay dung lop student
class student(person):
    def __init__(self, name, age, year):
        super().__init__(name, age)
        self.graduation = year
    def infor(self):
        super().infor()
        print("Graduation year: ",
self.graduation)
p1 = student("Phan Xuan Vong", 35, 2000)
p1.infor()
```

☞ Kết quả:

```
Name:  Phan Xuan Vong
Age:  35
Graduation year:  2000
```

7. So sánh hai đối tượng kiểu lớp

Các kiểu dữ liệu như kiểu số (int), kiểu chuỗi (str), kiểu danh sách (list), ... có hỗ trợ phép so sánh giữa hai đối tượng cùng kiểu. Đối với kiểu dữ liệu lớp cũng vậy, Python cho phép xây dựng phương thức so sánh hai đối tượng cùng một lớp.

Python hỗ trợ các phương thức đặc biệt để xây dựng các phép toán so sánh.

Kí hiệu phép toán	Phương thức thực hiện
<	__lt__(self, other)
>	__gt__(self, other)
<=	__le__(self, other)
>=	__ge__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)

Ví dụ 10. Xây dựng lớp person có phép toán so sánh và sử dụng phép toán so sánh để sắp xếp một danh sách các đối tượng.

Chương trình:

```
class person():
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def infor(self):
            print("Name: ",self.name)
            print("Age: ",self.age)
    def __lt__(self, other):
        return self.age < other.age
p1 = person("Vong", 35)
p2 = person("Yen", 30)
p3 = person("Duc", 10)
p4 = person("Vy", 6)
ds = [p1, p2, p3, p4]
ds.sort()
for x in ds:
    x.infor()
```

☞ Kết quả:

```
Name:  Vy
Age:  6
Name:  Duc
Age:  10
Name:  Yen
Age:  30
Name:  Vong
Age:  35
```

8. Xây dựng phép toán trên kiểu dữ liệu lớp

Python cũng cho phép xây dựng các phép toán trên kiểu dữ liệu lớp qua các phương thức đặc biệt.

Kí hiệu phép toán	Phương thức thực hiện
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
//	__floordiv__(self, other)
%	__mod__(self, other)
**	__pow__(self, other)

Ví dụ 11. Xây dựng lớp vector có các phép toán cộng, trừ và tích vô hướng.

Chương trình:

```
class vector():
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return self.x + other.x, self.y + other.y
    def __sub__(self, other):
        return self.x - other.x, self.y - other.y
    def __mul__(self, other):
        return self.x * other.x + self.y * other.y
u = vector(1, 2)
v = vector(3, 4)
```

```
a = u + v
b = u - v
s = u * v
print("Tong hai vector: ", a)
print("Hieu hai vector: ", b)
print("Tich vo huong:   ", s)
```

☞ Kết quả:

```
Tong hai vector:  (4, 6)
Hieu hai vector:  (-2, -2)
Tich vo huong:    11
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Lớp hình chữ nhật

Xây dựng một lớp có tên Rectange lưu thông tin về một hình chữ nhật. Có thuộc tính lưu chiều dài và chiều rộng. Có phương thức tính chu vi và tính diện tích hình chữ nhật.

Bài tập 2. Sắp xếp các danh sách các đối tượng hình chữ nhật

Với đối tượng Rectange được xây dựng ở bài tập 1, kế thừa và bổ sung thêm phương thức so sánh hai đối tượng theo tiêu chí: Đối tượng A gọi là nhỏ hơn đối tượng B nếu thỏa mãn một trong hai điều kiện:

- Diện tích của A nhỏ hơn diện tích của B.

- Diện tích của A bằng diện tích của B, chu vi của A nhỏ hơn chu vi của B.

Nhập một danh sách các đối tượng và đưa ra danh sách các đối tượng sau khi sắp xếp theo tiêu chí trên.

Bài tập 3. Tổ chức lời giải theo lớp

Xét bài toán: Cho dãy số a[0], a[1], .. , a[n-1]. Hãy thực hiện các công việc.

- Tính tổng các phần tử của dãy số.

- Tính phần tử nhỏ nhất.

- Tính phần tử lớn nhất.

- Đưa ra tất cả các bộ hai phần tử có tổng bằng 0.

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Lớp hình chữ nhật

Ý tưởng thuật toán

Sử dụng định nghĩa xây dựng lớp Rectange gồm hai thuộc tính chiều dài và chiều rộng.

Phương thức tính chu vi bằng tổng 4 cạnh.

Phương thức tính diện tích bằng tích 2 cạnh dài và rộng.

Chương trình

```
class Rectange():
    def __init__(self, dai, rong):
        self.dai = dai
        self.rong = rong
    def Chuvi(self):
        return 2*(self.dai + self.rong)
    def DienTich(self):
        return self.dai * self.rong
rect1 = Rectange(10, 5)
print("Chu vi: ", rect1.Chuvi())
print("DienTich: ",rect1.DienTich())
```

Bài tập 2. Sắp xếp các danh sách các đối tượng hình chữ nhật

Ý tưởng thuật toán

Xây dựng lớp RectangeSort được kế thừa từ lớp Rectange.

Xây dựng phương thức đặc biệt `__lt__(self, other)` để cài đặt phép toán so sánh hai đối tượng hình chữ nhật RectangeSort.

Chương trình

```
class Rectange():
    def __init__(self, dai, rong):
        self.dai = dai
        self.rong = rong
    def Chuvi(self):
        return 2*(self.dai + self.rong)
    def DienTich(self):
        return self.dai * self.rong
#Xay dung lop RectangeSort
class RectangeSort(Rectange):
    def __init__(self, dai, rong):
        super().__init__(dai, rong)
    def __lt__(self, other):
        dt_self = self.dai * self.rong
        cv_self = 2*(self.dai + self.rong)
        dt_other = other.dai * other.rong
        cv_other = 2*(other.dai + other.rong)
        if dt_self < dt_other:
            return True
        if dt_self == dt_other and cv_self < cv_other:
            return True
        return False
n = int(input("Nhap so hinh chu nhat: "))
print("Nhap chieu dai va chieu rong cua ", n, "hinh
chu nhat:")
ListRec = []
```

```
for i in range(n):
    dai = int(input())
    rong = int(input())
    ListRec.append(RectangeSort(dai, rong))
ListRec.sort()
print("Danh sach hinh chu nhat sau khi sap xep:")
for x in ListRec:
    print("Dien tich: ",x.DienTich(), " Chu vi:
",x.Chuvi())
```

Bài tập 3. Tổ chức lời giải theo lớp

Xét bài toán: Cho dãy số a[0], a[1], .. , a[n-1]. Hãy thực hiện các công việc.

- Tính tổng các phần tử của dãy số.
- Tính phần tử nhỏ nhất.
- Tính phần tử lớn nhất.
- Đưa ra tất cả các bộ hai phần tử có tổng bằng 0.

Ý tưởng thuật toán

Xây dựng một lớp có các phương thức thực hiện các công việc:

Phương thức `getsum()`, trả về tổng các số hạng của dãy.

Phương thức `getmin()`, trả về giá trị nhỏ nhất của dãy.

Phương thức `getmax()`, trả về giá trị lớn nhất của dãy.

Phương thực `pairsumtozero()`, trả về tất cả các cặp phần tử của dãy có tổng bằng 0.

Chương trình:

```
class solutionSeq():
    def __init__(self, list):
        self.list = list
    def getsum(self):
        s = 0
```

```
        for x in self.list:
            s += x
        return s
    def getmin(self):
        return min(self.list)
    def getmax(self):
        return max(self.list)
    def pairsumtozero(self):
        n = len(self.list)
        list_res = []
        for i in range(n-1):
            for j in range(i+1, n):
                if self.list[i] + self.list[j] == 0:
                    list_res.append((self.list[i],
self.list[j]))
        return list_res
seq = solutionSeq([1, 2, -1, 0, -2, 3])
print("Tong cac phan tu: ", seq.getsum())
print("Phan tu nho nhat: ", seq.getmin())
print("Phan tu lon nhat: ", seq.getmax())
print("Tong phan tu bang 0: ", seq.pairsumtozero())
```

☞ Kết quả:

```
Tong cac phan tu:  3
Phan tu nho nhat:  -2
Phan tu lon nhat:  3
Tong phan tu bang 0:  [(1, -1), (2, -2)]
```

CHỦ ĐỀ 19. MODUNLES VÀ PACKAGES

A. KIẾN THỨC CƠ BẢN

I. Modunle

Modunle trong Python là tập hợp các hàm, các lớp, các biến ..., và được tổ chức lưu trữ trên một tệp dữ liệu có phần mở rộng là **.py** (giống như một chương trình nguồn Python). Modunle giống như một thư viện được xây dựng sẵn các hàm, các chương trình, các lớp, các biến..., và có thể sử dụng chúng trong các chương trình khác. Python cung cấp rất nhiều modunle đã viết sẵn và cũng cho phép người lập trình viết những modunle của riêng mình, để sử dụng cho nhiều chương trình khác nhau.

1. Xây dựng một modunle

Ví dụ 1. Một modunle đơn giản gồm một hàm và một biến.

```
def hello():
    print("Hello world!")
langage = "Programming Python!"
```

Lưu chương trình với tên là **modunle1.py**. Chú ý là, khi thực hiện chương trình trên, ta không nhận được kết quả nào.

2. Sử dụng một modunle

Để sử dụng các hàm, các biến trong một modunle nào đó thì ta cần sử dụng câu lệnh **import** với cú pháp:

```
import <tên modunle>
```

Ví dụ 2. Chương trình sử dụng modunle1 được xây dựng ở trên.

```
import modunle1
print("Su dung ham hello trong modunle1:")
modunle1.hello()
print("Su dung bien langage trong modunle1:")
print(modunle1.langage)
```

☞ Kết quả:

```
Su dung ham hello trong modunle1:
Hello world!
Su dung bien langage trong modunle1:
Programming Python!
```

Như vậy, để sử dụng một hàm, hoặc một biến trong một modunle, ta sử dụng cú pháp:

```
<tên modunle>.<tên hàm>
<tên modunle>.<tên biến>
```

Câu lệnh `from <tên modunle> import`

Để sử dụng trực tiếp các hàm, các biến của một modunle nào đó, ta sử dụng câu lệnh **`from <tên modunle> import`** [danh sách các hàm, các biến].

Ví dụ 3. Chương trình sử dụng trực tiếp các hàm và biến của modunle1.

```
from modunle1 import hello, langage
print("Su dung ham hello trong modunle1:")
hello()
print("Su dung bien langage trong modunle1:")
print(langage)
```

☞ Kết quả:

```
Su dung ham hello trong modunle1:
Hello world!
Su dung bien langage trong modunle1:
Programming Python!
```

❖ ***Chú ý***

Câu lệnh: **`from <tên modunle> import *`** *sẽ nạp tất cả các hàm, các biến, các lớp,... có trong* **`<tên modunle>`**.

Như vậy, câu lệnh **`from modunle1 import hello, langage`**

Ta có thể viết cách khác **`from modunle1 import *`**

3. Sử dụng modunle trong các thư mục khác nhau

Ở các ví dụ trên, modunle1.py và các chương trình nguồn đều được lưu trong cùng một thư mục. Vậy nếu một modunle1.py và chương trình lưu trong hai thư mục khác nhau thì như thế nào? Câu trả lời là, Python sẽ không nhận biết được modunle1.py và sẽ không cho thực hiện chương trình.

Thêm đường dẫn

Giả sử modunle1.py được lưu trong thư mục C:/PythonModunle, các chương trình nguồn trong ví dụ 1, 2, 3 được lưu trong thư mục C:/PythonCode. Để chương trình trong ví dụ 1, 2, 3 sử dụng được modunle1.py, ta cần thêm đường dẫn theo cú pháp.

```
import sys
sys.path.append([Đường dẫn đến thư mục chứa modunle])
```

Ví dụ 4. Chương trình:

```
import sys
sys.path.append("C:/PythonModunle")
from modunle1 import hello, langage
print("Su dung ham hello trong modunle1:")
hello()
print("Su dung bien langage trong modunle1:")
print(langage)
```

Kết quả:

```
Su dung ham hello trong modunle1:
Hello world!
Su dung bien langage trong modunle1:
Programming Python!
```

II. Package

Mỗi package là một thư mục chứa các modunle, các package con. Trong mỗi package cần phải có một tệp **__init__.py**. Tên của thư mục cũng chính là tên của package.

1. Xây dựng một package

Bước 1. Tạo một thư mục có tên **package1**, đây cũng là tên của package cần tạo.

Bước 2. Tạo hai modunle có tên **flowers.py** và **sumlist.py** lưu trong thư mục package1.

Chương trình flowers.py

```
class flowerlist ():
    def __init__(self):
        self.listflower = ['Hoa Hong', 'Hoa hue', 'Hoa Lan']
    def printflower(self):
        for x in self.listflower:
            print(x)
```

Chương trình sumlist.py

```
def sum_list(List):
s = 0
for  x in List:
    s = s + x
return s
```

Bước 3. Xây dựng chương trình con **__init__.py** và lưu chúng trong thư mục package1.

```
#import class flowerlist from modunle flowere.py
from flowers import flowerlist
#import function sum_list from modunle sumlist.py
from sumlist import sum_list
```

Như vậy, ta có 3 tệp **flowers.py, sumlist.py, __init__.py** trong cùng một thư mục package1, đây chính là một package cần tạo.

2. Sử dụng một package

Chương trình sau sử dụng package1 được tạo ở trên. Giả sử thư mục package1 có đường dẫn: "C:\MyPackage\package1".

```
import sys
sys.path.append("C:\MyPackage\package1")
from flowers import flowerlist
from sumlist import sum_list
#Tao mot doi tuong dung class flowerlist trong package1
f1 = flowerlist()
f1.printflower()
#Su dung ham sum_list de tinh tong cac phan tu trong list
list = [1, 2, 3]
print("Tong cac phan tu: ",sum_list(list))
```

☞ Kết quả:

```
Hoa hong
Hoa hue
Hoa lan
Tong cac phan tu:  6
```

III. Ưu điểm của modunle và package

Ta nhận thấy, khi chương trình lớn, tức là chương trình cần giải quyết nhiều yêu cầu, nhiều công việc, có nhiều dữ liệu cần được lưu trữ và xử lý, thì việc tổ chức thành các modunle, package đem lại sự khoa học và hiệu quả. Các vấn đề được thiết kế theo nhóm có liên quan đến nhau và lưu trong các modunle, nhiều modunle lưu trong một package. Đối với mỗi modunle, package có thể giao cho một nhóm người thực hiện. Các modunle khác nhau thì có thể giao cho các nhóm người thực hiện khác nhau, điều này làm tăng hiệu quả công việc. Hơn nữa khi tổ chức thành các modunle, việc sửa đổi, nâng cấp được thực hiện trong các modunle đó, hệ thống không bị thay đổi. Một trong những điều làm nên sự ưu việt của Python đó là sự phong phú của các modunle, package được nhiều nhà lập trình nổi tiếng viết và chia sẻ rộng rãi cho cộng đồng.

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Xây dựng modunle có tên **geometry.py** gồm các lớp:

Lớp **`Circle`**`()` *mô tả đường tròn gồm các thuộc tính và phương thức.*

Thuộc tính:

- Tọa độ tâm (x; y).

- Bán kính `r` *của đường tròn.*

Phương thức:

- `getPerimeter()` *trả về chu vi đường tròn.*

- `getArea()` *trả về diện tích hình tròn.*

- `isIn(a,b)` *trả về True nếu điểm (a, b) nằm trong đường tròn, ngược lại trả về False.*

- `isOn(a,b)` *trả về True nếu điểm (a, b) nằm trên đường tròn, ngược lại trả về False.*

- `isIn(a,b)` *trả về True nếu điểm (a, b) nằm trong đường tròn, ngược lại trả về False.*

- `isOut(a,b)` *trả về True nếu điểm (a, b) nằm ngoài đường tròn, ngược lại trả về False.*

Lớp **`Square`**`()` *mô tả hình vuông gồm các thuộc tính và phương thức.*

Thuộc tính: độ dài cạnh a.

Phương thức:

- `getPerimeter()` *trả về chu vi hình vuông.*

- `getArea()` *trả về diện tích hình vuông.*

Viết chương trình có sử dụng modunle geometry.py.

Bài tập 2. Xây dựng modunle có tên **number.py** gồm các chương trình con:

Chương trình con `gcd(a,b)` *trả về ước chung lớn nhất của a và b.*

Chương trình con `lcm(a,b)` *trả về bội chung nhỏ nhất của a và b.*

Chương trình con `sumdivisor(n)` *trả về tổng các ước của n.*

Viết chương trình có sử dụng modunle number.py.

Bài tập 3. *Xây dựng một package có tên là mypackage gồm các modunle* geometry.py và number.py. *Viết chương trình có sử dụng package mypackage.*

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Xây dựng modunle có tên **geometry.py**

Ý tưởng thuật toán

Lớp ***Circle***():

- Phương thức *getPerimeter*() trả về chu vi bằng $2 * 3.14 * r$.

- Phương thức *getArea*() trả về diện tích bằng $3.14 * r * r$.

- Phương thức $isIn(a, b)$ trả về *True* nếu $(x - a) ** 2 + (y - b) ** 2 < r * r$.

thuộc đường tròn, ngược lại trả về *False*.

- Phương thức $isOn(a, b)$ trả về *True* nếu $(x - a) ** 2 + (y - b) ** 2 == r * r$.

thuộc đường tròn, ngược lại trả về *False*.

- Phương thức $isOut(a, b)$ trả về *True* nếu $(x - a) ** 2 + (y - b) ** 2 > r * r$.

thuộc đường tròn, ngược lại trả về *False*.

Lớp ***Square***():

- Phương thức *getPerimeter*() trả về chu vi bằng $4 * a$.

- Phương thức *getArea*() trả về diện tích bằng $a * a$.

Chương trình geometry.py

```
class Circle():
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r
    def getPerimeter(self):
```

```
        return 2*3.14*self.r
    def getArea(self):
        return 3.14*self.r**2
    def isIn(self, a, b):
        u = (self.x - a)**2 + (self.y - b)**2
        return(u < self.r**2)
    def isOn(self, a, b):
        u = (self.x - a)**2 + (self.y - b)**2
        return(u == self.r**2)
    def isOut(self, a, b):
        u = (self.x - a)**2 + (self.y - b)**2
        return(u == self.r**2)
class Square():
    def __init__(self, a):
        self.a = a
    def getPerimeter(self):
        return 4*self.a
    def getArea(self):
        return self.a*self.a
```

Chương trình sử dụng modunle geometry.py

```
from geometry import Circle, Square
#Tao mot doi tuong duong tron tam (1, 2), r = 4
c1 = Circle(1, 2, 4)
print("Chu vi: ", c1.getPerimeter())
print("Dien tich: ", c1.getArea())
#Kiem tra vi tri tuong doi cua diem (3, 5) voi
c1
```

```
if c1.isIn(3, 5):
    print("Nam trong duong tron: ")
if c1.isOn(3, 5):
    print("Nam tren duong tron: ")
if c1.isOut(3, 5):
    print("Nam ngoai duong tron: ")
#Tao mot doi tuong hinh vuong
sq1 = Square(4)
print("Chu vi: ",sq1.getPerimeter())
print("Dien tich: ",sq1.getArea())
```

Bài tập 2. Xây dựng modunle có tên **number.py**

Ý tưởng thuật toán

- Chương trình con $gcd(a, b)$: Sử dụng thuật toán Euclide để tìm ước chung lớn nhất của hai số nguyên dương.

- Chương trình con $lcm(a, b)$ trả về kết quả $(a * b)//(gcd(a, b))$.

- Chương trình con $sumdivisor(n)$:

Đặt s = 0.

Duyệt tất cả các số x = 1, 2, .., $\sqrt{n}$. Nếu n%x == 0 thì s = s + x; y = n//x; nếu y ! =x thì s = s + y.

Kết quả tổng các ước của n bằng s.

Chương trình number.py.

```
def gcd(a, b):
    while b > 0:
        r = a%b;
        a = b
        b = r
    return a
def lcm(a, b):
    u = a*b//gcd(a,b)
```

```
    return u
def sumdivisor(n):
    i = 1
    s = 0
    while i*i <= n:
        if n%i == 0:
            s += i
            j = n//i
            if i != j:
                s +=j
        i +=1
    return s
```

Chương trình có sử dụng modunle number.py.

```
from number import gcd, lcm, sumdivisor
print("Nhap hai so nguyen duong")
a = int(input("a = "))
b = int(input("b = "))
print("UCLN: ", gcd(a, b))
print("BCNN: ", lcm(a, b))
n = int(input("n = "))
print("Tong cac uoc cua n: ",sumdivisor(n))
```

Bài tập 3. Xây dựng một package có tên là mypackage gồm các modune geometry.py và number.py.

Bước 1. Tạo một thư mục có tên **mypackage.**

Bước 2. Copy hai modunle `geometry.py` và `number.py` vào trong thư mục mypackage.

Bước 3. Viết chương trình **`__init__.py`** và lưu trong thư mục mypackage.

```
from number import gcd, lcm, sumdivisor
```

```
from geometry import Circle, Square
```

Như vậy mypackage chứa ba tệp `geometry.py`, `number.py`, `__init__.py`.

Chương trình có sử dụng **mypackage**.

```
import sys
#Them duong dan den thu muc mypackage
sys.path.append("C:\MyPackage\mypackage")
import mypackage
#Tao doi tuong hinh vuong
sq = mypackage.Square(20)
print("Chu vi: ",sq.getPerimeter())
print("Dien tich: ",sq.getArea())
n = 100
print("Tong cac uoc cua n:
",mypackage.sumdivisor(n))
```

❖ ***Chú ý***

Để sử dụng câu lệnh **from mypackage import *** *để nạp tất cả các hàm, lớp, biến có trong mypackage, ta cần sửa tệp* **__init__.py** *như sau:*

```
from number import gcd, lcm, sumdivisor
from geometry import Circle, Square
__all__ = ["gcd","lcm","sumdivisor", "Circle",
"Square"]
```

CHỦ ĐỀ 20. MODUNLE QUEUE VÀ CẤU TRÚC DỮ LIỆU HÀNG ĐỢI, NGĂN XẾP, HÀNG ĐỢI ƯU TIÊN

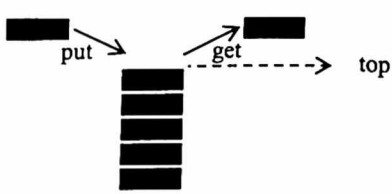
A. KIẾN THỨC CƠ BẢN

1. Khái niệm cấu trúc dữ liệu hàng đợi - queue

Hàng đợi (queue) là danh sách các phần tử, trong đó có hỗ trợ hai phép toán cơ bản: Phép thêm một phần tử vào danh sách (put) được thực hiện tại một đầu và phép lấy ra một phần tử khỏi danh sách (get) được thực hiện ở đầu còn lại của danh sách. Phép thêm vào một phần tử và phép lấy ra một phần tử theo cách trên còn gọi là nguyên lý vào trước ra trước (*First In First Out - FIFO*).

2. Khái niệm cấu trúc dữ liệu ngăn xếp – stack

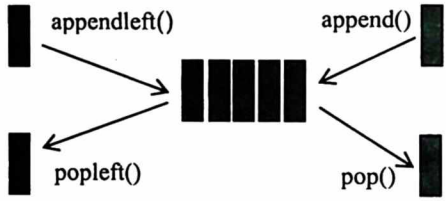
Ngăn xếp là một danh sách các phần tử, trong đó có hỗ trợ hai phép toán cơ bản: Phép thêm một phần tử vào danh sách (`put`) và phép lấy ra một phần tử khỏi danh sách (`get`) được thực hiện tại một đầu của danh sách. Phép thêm một phần tử vào danh sách và phép lấy ra một phần tử khỏi danh sách theo cách trên được gọi là nguyên lý vào sau ra trước (*Last In First Out - LIFO*).



Phần tử được thêm vào sau cùng được gọi là phần tử ở đỉnh danh sách (top).

3. Khái niệm cấu trúc dữ liệu hàng đợi hai đầu - deque

Hàng đợi hai đầu (double ended queue - deque) là danh sách các phần tử, trong đó có hỗ trợ hai phép toán cơ bản: Phép thêm một phần tử vào danh sách và phép lấy ra một phần tử khỏi danh sách. Hai phép toán này đều có thể được thực hiện tại hai đầu của danh sách.



- Phép **append(x)** thêm phần tử x vào đầu bên phải của danh sách.

- Phép **pop()** lấy phần tử ở đầu bên phải ra khỏi danh sách.

- Phép **appendleft(x)** thêm phần tử x vào đầu bên trái của danh sách.

- Phép **popleft()** lấy phần tử ở đầu bên trái ra khỏi danh sách.

4. Khái niệm về hàng đợi ưu tiên

Hàng đợi ưu tiên (Priority queue) là danh sách các phần tử, mỗi phần tử có thể so sánh được. Phần tử a '*nhỏ hơn*' phần tử b thì ta nói phần tử a có độ '*ưu tiên*' (*priority*) nhỏ hơn phần tử b. Hàng đợi ưu tiên hỗ trợ hai phép toán cơ bản. Phép thêm vào một phần tử và phép lấy ra phần tử có độ ưu tiên nhỏ nhất. Hàng đợi ưu tiên thường được tổ chức theo cấu trúc heap, các phép toán thêm vào một phần tử và lấy ra một phần tử đều có độ phức tạp thời gian là $O(log_2 n)$ với n là số phần tử trong hàng đợi.

5. Modunle queue trong Python

Python cung cấp modunle queue cho phép thực hiện các cấu trúc dữ liệu hàng đợi, ngăn xếp và hàng đợi hai đầu.

a) Cấu trúc dữ liệu hàng đợi - queue

Để sử dụng cấu trúc dữ liệu hàng đợi trong modunle queue, ta thực hiện:

```
import sys
import os
import argparse
import queue
```

Ta cũng có thể viết trên một dòng:

```
import sys, os, argparse, queue
```

- **Khai báo biến kiểu dữ liệu hàng đợi**

<tên biến> = queue.Queue()

Nếu hạn chế số phần tử trong hàng đợi, ta khai báo:

<tên biến> = **queue.Queue(**<Số phần tử tối đa>**)**

- **Thêm một phần tử vào hàng đợi**

<tên biến>**.put(**<phần tử thêm vào>**)**

- **Lấy một phần tử ra khỏi hàng đợi**

<tên biến>**.get()**

- **Hàm kiểm tra hàng đợi có rỗng**

<tên biến>**.empty()**

Hàm trả về True nếu hàng đợi rỗng, ngược lại trả về False.

- **Hàm kiểm tra hàng đợi đầy**

<tên biến>**.full()**

Hàm trả về True nếu hàng đợi đầy, ngược lại trả về False.

- **Hàm lấy số phần tử trong hàng đợi**

<tên biến>**.qsize()**

Hàm trả về số phần tử trong hàng đợi.

Ví dụ 1. Chương trình:

```
import sys, os, argparse, queue
my_queue = queue.Queue()
```

```
my_queue.put(1)
my_queue.put(2)
my_queue.put("Python")
print("So phan tu trong hang doi:",
my_queue.qsize())
print("Cac phan tu trong hang doi")
while not my_queue.empty():
    x = my_queue.get()
    print(x)
```

☞ Kết quả:

```
So phan tu trong hang doi: 3
Cac phan tu trong hang doi
1
2
Python
```

Ví dụ 2. Chương trình:

```
import sys, os, argparse, queue
my_queue = queue.Queue(5) #So phan tu toi da =
5
for i in range(10):
    if my_queue.full():
        print("Hang doi da day")
        break
    else:
        my_queue.put(i)
print("Cac phan tu trong hang doi")
while not my_queue.empty():
    x = my_queue.get()
    print(x)
```

☞ Kết quả:

```
Hang doi da day
Cac phan tu trong hang doi
0
1
2
3
4
```

b) Cấu trúc dữ liệu ngăn xếp - stack

Khai báo biến dữ liệu kiểu ngăn xếp

<tên biến> = **queue.LifoQueue()**

<tên biến> = **queue.LifoQueue(**<Số phần tử tối đa>**)**

Các hàm với kiểu dữ liệu ngăn xếp

Kiểu dữ liệu ngăn xếp có các hàm tương tự như kiểu dữ liệu hàng đợi.

Ví dụ 3. Chương trình:

```
import sys, os, argparse, queue
my_stack = queue.LifoQueue()
my_stack.put(1)
my_stack.put(2)
my_stack.put("Python")
print("So phan tu trong ngan
xep:",my_stack.qsize())
print("Cac phan tu:")
while not my_stack.empty():
   x = my_stack.get()
   print(x)
```

☞ Kết quả:

```
So phan tu trong ngan xep: 3
Cac phan tu:
Python
2
1
```

Ví dụ 4. Chương trình:

```
import sys, os, argparse, queue
my_stack = queue.LifoQueue(5)
for i in range(10):
   if my_stack.full():
      print("Ngan xep day!")
      break
   else:
      my_stack.put(i)
print("So phan tu trong ngan
xep:",my_stack.qsize())
print("Cac phan tu:")
while not my_stack.empty():
   x = my_stack.get()
   print(x)
```

☞ Kết quả:

```
Ngan xep day!
So phan tu trong ngan xep: 5
Cac phan tu:
4
3
2
1
0
```

c) Cấu trúc dữ liệu hàng đợi hai đầu - deque

- Khai báo biến dữ liệu kiểu hàng đợi hai đầu

<tên biến> = **queue.deque()**

<tên biến> = **queue.deque(**<Số phần tử tối đa>**)**

- Thêm một phần tử vào đầu bên phải deque

<tên biến>.**append**(<Phần tử>)

- Thêm một phần tử vào đầu bên trái deque

<tên biến>.**appendleft**(<Phần tử>)

- Lấy một phần tử ở đầu bên phải của deque

<tên biến>.**pop()**

- Lấy một phần tử ở đầu bên trái của deque

<tên biến>.**popleft()**

Ví dụ 5. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque()
my_deque.append(1)
my_deque.append(2)
my_deque.append(3)
print(my_deque)
my_deque.appendleft("Them trai 1")
my_deque.appendleft("Them trai 2")
print(my_deque)
```

☞ Kết quả:

```
deque([1, 2, 3])
deque(['Them trai 2', 'Them trai 1', 1, 2, 3])
```

Ví dụ 6. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque()
```

```
for i in range(10):
    my_deque.append(i)
print(my_deque)
print("Lay phan tu ben phai:",my_deque.pop())
print("Lay phan tu ben
trai:",my_deque.popleft())
print(my_deque)
```

☞ Kết quả:

```
deque([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
Lay phan tu ben phai: 9
Lay phan tu ben trai: 0
deque([1, 2, 3, 4, 5, 6, 7, 8])
```

- Tạo deque từ một danh sách

<tên biến> = **queue.deque**([danh sách các phần tử])

Ví dụ 7. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque([1, 2, 3,4])
print(my_deque)
```

☞ Kết quả:

```
deque([1, 2, 3, 4])
```

- Thêm danh sách các phần tử vào deque

<tên biến>.**extend**([danh sách các phần tử])

Thêm các phần tử của [danh sách các phần tử] vào bên phải của <tênbiến>.

<tên biến>.**extendleft**([danh sách các phần tử])

Thêm các phần tử của [danh sách các phần tử] vào bên trái của <tên biến>.

Ví dụ 8. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque([1, 2, 3, 4])
my_deque.extend(['phai', 'phai'])
my_deque.extendleft(['trai', 'trai'])
print(my_deque)
```

☞ Kết quả:

```
deque(['trai', 'trai', 1, 2, 3, 4, 'phai', 'phai'])
```

- Xóa tất cả các phần tử thuộc deque

<tên biến>.**clear()**

- Đảo ngược các phần tử trong deque

<tên biến>.**reverse()**

Ví dụ 9. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque([1, 2, 3, 4])
print("deque ban dau:",my_deque)
my_deque.reverse()
print("deque dao nguoc:", my_deque)
my_deque.clear()
print("deque sau khi xoa:", my_deque)
```

☞ Kết quả:

```
deque ban dau: deque([1, 2, 3, 4])
deque dao nguoc: deque([4, 3, 2, 1])
deque sau khi xoa: deque([])
```

- **Sự giống nhau giữa deque và list**

Từ các câu lệnh **print(my_deque)** ta có thấy rằng, deque có nhiều điểm giống list. Deque được tổ chức dưới dạng một list có hỗ trợ một số phép toán như trên. Các phần tử trong deque được đánh chỉ số từ 0, 1, .. (từ trái sang phải).

Deque không hỗ trợ phép toán kiểm tra rỗng, nhưng ta có thể biết deque rỗng bằng cách tính số phần tử thông qua hàm **len()**.

Do vậy phần tử bên trái của my_deque là my_deque[0] và phần tử bên phải của my_deque[len(my_deque) - 1].

Ví dụ 10. Chương trình:

```
import sys, os, argparse, queue
my_deque = queue.deque([1, 2, 3, 4])
n = len(my_deque)
print("So phan tu:", n)
print("Cac phan tu:")
for x in my_deque:
   print(x, end = ' ')
print()
for i in range(n):
   print(my_deque[i], end = ' ')
```

☞ Kết quả:

```
So phan tu: 4
Cac phan tu:
1 2 3 4
1 2 3 4
```

d) Cấu trúc dữ liệu hàng đợi ưu tiên

- Khai báo biến kiểu dữ liệu hàng đợi ưu tiên

<tên biến> = **queue.PriorityQueue()**

<tên biến> = **queue.PriorityQueue(**[Số phần tử tối đa]**)**

- Thêm một phần tử vào hàng đợi ưu tiên

<tên biến>.**put(x)**

- Lấy một phần tử ra khỏi hàng đợi ưu tiên

<tên biến>.**get()**

- Hàm kiểm tra hàng đợi ưu tiên rỗng

<tên biến>.**empty()**

Hàm trả về True nếu hàng đợi rỗng, ngược lại trả về False.

- Hàm kiểm tra hàng đợi ưu tiên đầy

<tên biến>.**full()**

Hàm trả về True nếu hàng đợi đầy, ngược lại trả về False.

- Hàm lấy số phần tử trong hàng đợi

<tên biến>.**qsize()**

Ví dụ 11. Chương trình:

```
import sys, os, argparse, queue
my_priority = queue.PriorityQueue()
my_priority.put(10)
my_priority.put(1)
my_priority.put(100)
while not my_priority.empty():
   print(my_priority.get())
```

☞ Kết quả:

```
1
10
100
```

Ví dụ 12. Chương trình:

```
import sys, os, argparse, queue
my_priority = queue.PriorityQueue()
my_priority.put((10, 'Toan'))
my_priority.put((9, 'Tin'))
my_priority.put((10, 'Ly'))
while not my_priority.empty():
   print(my_priority.get())
```

☞ Kết quả:

```
(9, 'Tin')
(10, 'Ly')
(10, 'Toan')
```

Ví dụ 13. Chương trình:

```
import sys, os, argparse, queue
my_priority = queue.PriorityQueue(5)
for i in range(10, 0, -1):
   if my_priority.full():
      print("my_priority is full.")
      break
   else:
      my_priority.put(i)
print("So phan tu:", my_priority.qsize())
while not my_priority.empty():
   print(my_priority.get())
```

☞ Kết quả:

```
my_priority is full.
So phan tu: 5
6
7
8
9
10
```

❖ **Hàng đợi ưu tiên của dãy các đối tượng**

Các phần tử trong hàng đợi ưu tiên có thể là các đối tượng lớp (class). Tuy nhiên, các đối tượng lớp này cần được định nghĩa phép so sánh giữa chúng.

Ví dụ 14. Xây dựng lớp đối tượng điểm gồm hai thuộc tính x, y. Ta nói điểm A có độ ưu tiên nhỏ hơn điểm B nếu:

Hoặc A.x + A.y > B.x + B.y

Hoặc A.x + A.y = B.x + B.y và A.x > B.x

Đưa vào hàng đợi ưu tiên một số đối tượng và sau đó đưa các đối tượng được bỏ vào ra khỏi hàng đợi.

Chương trình

```
import sys, os, argparse, queue
class point():
   def __init__(self, x, y):
      self.x = x
      self.y = y
   def __lt__(self, other):
      if self.x + self.y > other.x + other.y:
         return True
      if self.x + self.y == other.x + other.y:
         return self.x > self.y
   def Print(self):
      print((self.x, self.y))
my_priority = queue.PriorityQueue()
my_priority.put(point(8,10))
my_priority.put(point(10,8))
my_priority.put(point(10,10))
my_priority.put(point(11,2))
while not my_priority.empty():
   x = my_priority.get()
   x.Print()
```

☞ Kết quả:

```
(10, 10)
(10, 8)
(8, 10)
(11, 2)
```

B. BÀI TẬP ÔN LUYỆN

Bài tập 1. Tìm số

Cho k chữ số khác nhau $a_0, a_1, .., a_{k-1}$ và hai số nguyên dương M_1, M_2. Hãy sử dụng các chữ số trong k chữ số trên để ghép (tạo) thành số X thỏa mãn:

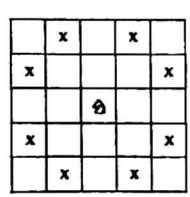
- *X chia hết cho M_1*
- *$X \leq M_2$*
- *X là số nhỏ nhất có thể.*

Một chữ số có thể được sử dụng nhiều lần để tạo nên số X. Trong trường hợp không có X thỏa mãn, chương trình đưa ra thông báo "Khong co X".

Ví dụ: Cho 3 chữ số 1, 2, 3; $M_1 = 6$, $M_2 = 100$. Số X ghép được là: 12.

Bài tập 2. Mã đi tuần nhanh nhất

Cho lưới ô vuông gồm m dòng và n cột. Các dòng được đánh chỉ số từ 0 đến m – 1 (từ trên xuống dưới), các cột đánh chỉ số từ 0 đến n – 1 (từ trái sang phải). Như vậy lưới gồm m × n ô vuông nhỏ. Ô vuông ở dòng i, cột j được gọi là ô (i, j). Ban đầu có một quân mã đặt tại ô (x, y), cần di chuyển quân mã đến ô (u, v) với số lần di chuyển là ít nhất.



Cách di chuyển của quân mã là: Tại ô (i,j), quân mã có thể di chuyển đến một trong 8 ô $(i-1,j-2)$, $(i-1,j+2)$, $(i+1,j-2)$, $(i+1,j+2)$, $(i-2,j-1)$, $(i-2,j+1)$, $(i+2,j-1)$, $(i+2,j+1)$, hình vẽ. Tất nhiên, các ô này không được ra ngoài lưới.

Viết chương trình nhập m,n,x,y,u,v với $m,n \leq 1000, 0 \leq x,u < m$; $0 \leq y,v < n$. Hãy tính số lần di chuyển ít nhất để đưa quân mã từ ô (x,y) đến ô (u,v). Nếu không thể di chuyển được thì thông báo 'Khong di chuyen duoc !'.

Bài tập 3. Chuỗi ngoặc đúng

Cho chuỗi kí tự gồm hai loại kí tự '(' và ')'. Chuỗi ngoặc đúng được định nghĩa:

- Chuỗi rỗng là một chuỗi ngoặc đúng.

- Nếu A là chuỗi đúng thì chuỗi tạo thành khi thêm '(' vào bên trái, thêm kí tự ')' vào bên phải thì ta được chuỗi ngoặc đúng. Tức là chuỗi (A) là chuỗi ngoặc đúng.

- Nếu A, B là hai chuỗi ngoặc đúng thì chuỗi ghép AB là chuỗi ngoặc đúng.

Ví dụ: Các chuỗi: (), ()()(), (()()) là các chuỗi ngoặc đúng. Các chuỗi (()), ()()) không phải là chuỗi ngoặc đúng.

Viết chương trình nhập vào một chuỗi gồm các kí tự '(' và ')'. Kiểm tra xem có phải là chuỗi ngoặc đúng hay không?

Bài tập 4. Bậc của chuỗi ngoặc đúng

Cho St là một chuỗi ngoặc đúng, bậc của chuỗi St được kí hiệu là `deg(St)`. Bậc của một chuỗi được định nghĩa:

- Chuỗi rỗng có bậc là 0.

- A là chuỗi ngoặc đúng, gọi C là chuỗi (A). Khi đó deg(C) = deg(A) + 1.

- A, B là hai chuỗi ngoặc đúng, gọi C là chuỗi AB. Khi đó deg(C) = max(deg(A), deg(B)).

Ví dụ: A = ()()(()), deg(A) = 2; B = ((()())), deg(B) = 3.

Viết chương trình nhập vào một chuỗi ngoặc đúng. Tính bậc của ngoặc nhập vào.

Bài tập 5. Xóa k chữ số

Cho số tự nhiên X có n ($n \leq 10^5$) chữ số. Hãy tìm cách xóa đi k chữ số ($k < n$) của X để các chữ số còn lại (vẫn giữ nguyên thứ tự) tạo thành số lớn nhất.

Ví dụ: X = 192637, k = 3. Ta xóa 3 chữ số: 1, 2, 3 để tạo thành số 967 là số lớn nhất có thể.

Bài tập 6. Giá trị lớn nhất trong đoạn gồm *k* số hạng liên tiếp của dãy số

Cho dãy số nguyên gồm n số hạng: $a_0, a_1, .., a_{n-1}$ và số $k < n$. Hãy tìm giá trị lớn nhất của các đoạn gồm k số hạng liên tiếp.

$a_0, a_1, .., a_{k-1}$;

$a_1, a_2, .., a_k$;

.....

$a_{n-k}, .., a_{n-1}$

Ví dụ, dãy: 1, 2, 3, 2, 5; k = 3. Ta có:

Giá trị lớn nhất của dãy: [1, 2, 3] là 3; [2, 3, 2] là 3; [3, 2, 5] là 5.

Bài tập 7. Trò chơi Bỏ vào - Lấy ra

Trò chơi Bỏ vào - Lấy ra được thực hiện trên các con số như sau:

Cho tập S gồm n nguyên $s_0, s_1, .., s_{n-1}$. Trò chơi được thực hiện k lần, mỗi lần gồm 2 bước:

Bước 1: Lấy ra số nhỏ nhất khỏi tập S, tạ gọi là x.

Bước 2: Thêm vào tập S số có giá trị bằng 2x.

Như vậy, sau mỗi lần chơi, tập S luôn có n số.

Yêu cầu: Hãy in ra các số x được lấy ra (theo tuần tự của lượt chơi).

Bài tập 8. Truy vấn trung vị

- *Khái niệm trung vị của một dãy*

Xét dãy gồm lẻ số hạng $a_0, a_1, .., a_{n-1}$, với n là số lẻ. Sắp xếp dãy theo thứ tự tăng dần (hoặc giảm dần). Phần tử có chỉ số $\frac{n}{2}$ được gọi là phần tử trung vị của dãy. Ví dụ, dãy: 1, 3, 2, 6, 8; sắp xếp dãy thì có: 1, 2, 3, 6, 8. Phần tử trung vị bằng 3.

Truy vấn trung vị

Cho dãy A ban đầu gồm một số hạng a_0. Tiến hành thực hiện n lần, mỗi lần gồm:

Bước 1: Thêm dãy A vào hai số hạng x và y.

Bước 2: Tìm phần tử trung vị của dãy A.

Viết chương trình thực hiện n lần trên.

Ví dụ: n = 3, A = [1].

Lần 1: Thêm 2, 3, A = [1, 2, 3], *trung vị là 2.*

Lần 2: Thêm 1, -1, A = [1, 2, 3, 1, -1], *sắp xếp* A = [-1, 1, 1, 2, 3], *trung vị là 1.*

Lần 3: Thêm 9, 9, A = [-1, 1, 1, 2, 3, 9, 9], *trung vị là 2.*

C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1. Tìm số

Ý tưởng thuật toán

Ta sắp xếp k chữ số $a_0, a_1, .., a_{k-1}$ theo thứ tự tăng dần để được $a_0 < a_1 < .. < a_{k-1}$.

Nhận thấy nếu $a_0 = 0$ thì X = 0 là số cần tìm.

Ta xét trường hợp $a_0 > 0$.

Để tìm X nhỏ nhất, ta sẽ lần lượt xét tập các số có 1 chữ số, có 2 chữ số, ... Trong mỗi tập có cùng số chữ số, ta sẽ xét các số từ nhỏ đến lớn. Để làm điều này, sẽ sử dụng hàng đợi để lần lượt tạo ra các số từ k chữ số đã cho.

Ví dụ: Cho 2 chữ số: 1, 3, các số lần lượt được tạo ra:

Lớp 1 chữ số: 1, 3.

Lớp 2 chữ số: 11, 13, 31, 33.

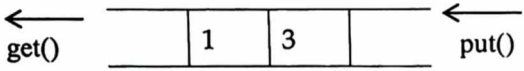
Lớp 3 chữ số: 111, 113, 131, 133, 311, 313, 331, 333.

...

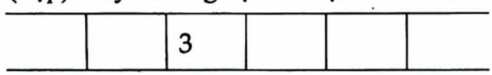
Các số được tạo ra theo thứ tự tăng dần, mỗi lần tạo ra một số, kiểm tra xem nếu số đó chia hết cho M_1 thì số đó chính là X cần tìm. Trong quá trình tạo số, gặp số lớn hơn M_2 thì sẽ kết thúc.

Sử dụng hàng đợi để tạo số:

Bước 1. (Khởi tạo). Cho k chữ số vào hàng đợi (lớp có 1 chữ số)



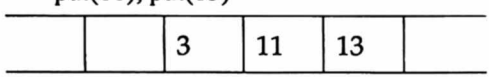
Bước 2. (Lặp). Lấy ra hàng đợi → Tạo số → Bỏ vào hàng đợi.

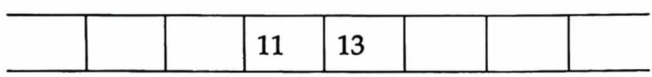


1 = get()

Tạo số: 11, 13

put(11), put(13)

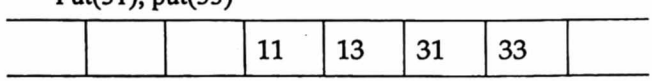




3 = get()

Tạo số: 31, 33

Put(31), put(33)



Như vậy ta đã tạo ra được tất cả các số có 2 chữ số (lớp 2 chữ số). Lớp có 3 chữ số được thực hiện tương tự.

Thao tác tạo ra số mới được thực hiện:

$u = get()$, k số mới được tạo thành: $u \times 10 + a_0$; $u \times 10 + a_1$; .. ; $u \times 10 + a_{k-1}$.

Chương trình

```
import sys, os, argparse, queue
my_queue = queue.Queue()
k  = int(input("Nhap so chu so:"))
```

```
M1 = int(input("Nhap M1:"))
M2 = int(input("Nhap M2:"))
print("Nhap",k,"chu so khac nhau:")
a = [int(input()) for i in range(k)]
a.sort()
if a[0] == 0:
   X = 0
   print("So can tim:",X)
   exit(0)
for x in a:
   my_queue.put(x)
while not my_queue.empty():
   x = my_queue.get()
   if x % M1 == 0 and x <= M2:
      X = x
      print("So can tim:",X)
      exit(0)
   for t in a:
      u = x*10 + t;
      if u <= M2:
         my_queue.put(u)
print("Khong co X")
```

Bài tập 2. Mã đi tuần nhanh nhất

Ý tưởng thuật toán

Ta sẽ lần lượt duyệt theo số bước di chuyển quân mã.

+ Tập các ô sau 0 bước di chuyển: $S_0 = \{(x, y)\}$. Nếu $(u, v) \in S_0$ thì sau 0 bước di chuyển thì đến được (u, v).

+ Tập các ô đến được sau ít nhất 1 bước di chuyển: S_1 = {(x – 1, y – 2), (x – 1, y + 2), (x+1, y – 2), (x+1, y+2), (x – 2, y – 1), (x – 2, y + 1), (x + 2, y – 1), (x+2, y+1) }.

Nếu (u, v) ∈ S_1 thì sau ít nhất 1 bước di chuyển thì đến được (u, v). Ngược lại, sau 1 bước không thể đến được ô (u, v).

+ Từ các ô thuộc S_1, ta tạo được các tập S_2 gồm các ô đến được sau ít nhất 2 bước di chuyển.

Nếu (u, v) ∈ S_2 thì sau 2 bước di chuyển thì đến được (u, v). Ngược lại, sau 2 bước không thể đến được ô (u, v). Và ta tiếp tục xây dựng tập S_3 gồm các đỉnh đến được sau ít nhất 3 bước.

Gọi S_k là tập các ô có thể di chuyển đến được sau ít nhất k bước. Ứng với mỗi k, kiểm tra (u, v) ∈ S_k? Nếu thuộc, ta có kết quả là cần ít nhất k bước để di chuyển đến ô (u, v), ngược lại, tiếp tục xây dựng S_{k+1}. Nếu tập S_{k+1} là rỗng thì kết luận không có cách di chuyển đến ô (u, v).

Sơ đồ mở rộng các tập: $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_k \rightarrow S_{k+1} \ldots$

Sử dụng danh sách hằng:

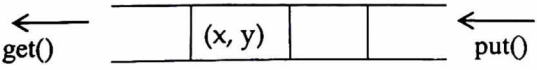
```
drow = [-1, -1, 1, 1, -2,-2,  2, 2]
dcol = [-2,  2,-2, 2, -1, 1, -1, 1]
```

Khi đó từ ô `(i, j)`, ta đến được 8 ô:

```
for t in range(8):
    (i+drow[t], j+dcol[t])
```

Sử dụng hàng đợi để mở rộng các tập:

Bước 1. (*Khởi tạo*) Cho ô (x, y) vào hàng đợi (S_0)



Bước 2. (*Lặp*). Lấy ra hàng đợi → Từ ô lấy ra, di chuyển đến các ô chưa đến → Bỏ vào hàng đợi.

(x, y) = **get()**

Từ ô (x, y), đến các ô (x–1, y–2), (x–1, y+2), (x+1, y–2), (x+1, y+2), (x–2, y–1), (x–2, y+1), (x+2, y–1), (x+2, y+1).

Nếu (u, v) thuộc tập các ô này, ta có lời giải, ngược lại put() vào hàng đợi và lặp lại.

put()

(x–1,y–2)	(x–1,y+2)	(x+1,y–2)	(x+1,y+2)	(x–2,y–1)	(x-2,y+1)	(x+2, y–1)	(x+2,y+1)

Để lưu số bước di chuyển ít nhất từ ô (x, y) đến một ô (i, j), ta sử dụng danh sách hai chiều d[i][j]. Như vậy d[x][y] = 0 và kết quả là d[u][v]. Ta cũng sử dụng d[i][j] để biết thông tin đã đến ô (i, j) hay chưa trong quá trình xây dựng các tập S_0, S_1, S_2, ... Nếu d[i][j] = 0 và (i, j) ! = (x, y) thì có nghĩa chưa đến ô (i, j), ngược lại là đã đến ô (i, j).

Chương trình

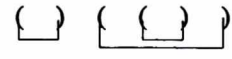
```
import sys, os, argparse, queue
my_queue = queue.Queue()
def Inboard(i, j):
   if 0 <= i < m and 0 <= j < n:
      return True
   else:
      return False
m  = int(input("Nhap so m <= 1000:"))
n  = int(input("Nhap so n <= 1000:"))
x  = int(input("Nhap x:"))
y  = int(input("Nhap y:"))
u  = int(input("Nhap u:"))
v  = int(input("Nhap v:"))
if (x, y) == (u, v):
   print("Khong can di chuyen!")
   exit(0)
```

```
drow = [-1, -1, 1, 1, -2,-2,  2, 2]
dcol = [-2,  2,-2, 2, -1, 1, -1, 1]
d = [[0 for i in range(n)] for j in range(m)]
my_queue.put((x, y))
while not my_queue.empty() and d[u][v] == 0:
   (xt, yt) = my_queue.get()
   for t in range(8):
      i = xt + drow[t]
      j = yt + dcol[t]
      if not Inboard(i, j): continue
      if (i, j) != (x, y) and d[i][j] == 0:
         d[i][j] = d[xt][yt] + 1
         my_queue.put((i,j))
if d[u][v] == 0:
   print("Khong di chuyen duoc.")
else:
   print("So buoc di chuyen it nhat:",d[u][v])
```

Bài tập 3. Chuỗi ngoặc đúng

Ý tưởng thuật toán

Ta nhận thấy, trong một chuỗi ngoặc đúng, mỗi kí tự đóng ')' đều tương ứng với một kí tự mở ')' duy nhất.



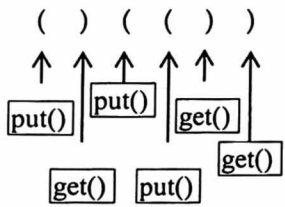
Hơn nữa, với mỗi ngoặc đóng sẽ tương ứng với một ngoặc mở kề trái nhất mà chưa tương ứng với ngoặc đóng nào. Điều này gợi ý cho ta sử dụng cấu trúc dữ liệu ngăn xếp để ghép cặp 'đóng' – 'mở'. Nếu trong quá trình ghép, nếu có một kí tự đóng ')' hoặc '(' không được ghép thì chuỗi không phải là chuỗi ngoặc đúng.

Để thực hiện việc ghép cặp 'đóng' – 'mở', ta lần duyệt các kí tự của chuỗi từ trái sang phải.

o Gặp ngoặc mở '(' thì đưa vào ngăn xếp.

o Gặp ngoặc đóng ')':

- Nếu ngăn xếp rỗng thì kết luận không phải chuỗi ngoặc đúng.
- Nếu ngăn xếp khác rỗng, thì lấy phần tử ở đỉnh ngăn xếp.



Chương trình

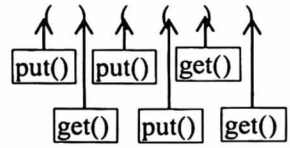
```
import sys, os, argparse, queue
my_stack = queue.LifoQueue()
str = input("Nhap chuoi de kiem tra: ")
for x in str:
   if x == '(':
      my_stack.put(x)
   else:
      if my_stack.empty():
         print("Khong phai chuoi ngoac dung")
         exit(0)
      else:
         my_stack.get()
if my_stack.empty():
   print("La chuoi ngoac dung")
else:
   print("Khong phai chuoi ngoac dung")
```

Bài tập 4. Bậc của chuỗi ngoặc đúng

Ý tưởng thuật toán

Ta nhận thấy bậc của chuỗi ngoặc đúng là số kí tự ngoặc mở nằm trong ngăn xếp khi đạt giá trị lớn nhất (lúc trong ngăn xếp có nhiều ngoặc mở nhất).

Ví dụ:



Số ngoặc mở trong ngăn xếp: 1 0 1 2 1 0

Như vậy, bậc của chuỗi ngoặc đúng trên bằng 2.

Chương trình

```
import sys, os, argparse, queue
my_stack = queue.LifoQueue()
str = input("Nhap chuoi ngoac dung: ")
deg = 0
for x in str:
   if x == '(':
      my_stack.put(x)
      deg = max(deg, my_stack.qsize())
   else:
      my_stack.get()
print("Bac cua chuoi bang:",deg)
```

Bài tập 5. Xóa k chữ số

Ý tưởng thuật toán

Ta viết X dưới dạng cấu tạo số $X = \overline{a_0 a_1 a_2 .. a_{n-1}}$, $(a_0 > 0)$.

Nhận thấy, số tạo thành sau khi xóa có $n - k$ chữ số (không đổi khi cho n và k). Do vậy, chữ số bên trái càng lớn thì được giá trị càng lớn.

Để được số lớn nhất, ta xét các chữ số từ trái sang phải, nếu gặp một số nhỏ hơn số kề sau thì xóa số đó. Lại tiếp tục thực hiện như vậy đến lúc:

- Hoặc đã xóa đủ k chữ số, khi đó các chữ số còn lại tạo thành số lớn nhất;

- Hoặc không có chữ số bên trái lại nhỏ hơn bên phải, tức là các chữ số (từ trái sang phải) sẽ giảm dần, lúc này ta sẽ xóa các số ở cuối bên phải cho đủ k chữ số.

Ví dụ 1. X = 15364, k= 3.

Lần xóa 1: X = 15364, ta có 1 < 5, xóa 1, X = 5364.

Lần xóa 2: X = 5364 , ta có 3 < 6, xóa 3, X = 564.

Lần xóa 3: X = 564, ta có 5 < 6, xóa 5, X = 64.

Vậy số lớn nhất đạt được sau khi xóa 3 chữ số là 64.

Ví dụ 2. X = 2876554, k = 3.

Lần xóa 1: X = 2876554, ta có 2 < 8, xóa 2, X = 876554.

Ta có các chữ số của X đều giảm dần, do vậy sẽ xóa 2 chữ số cuối bên phải để được số lớn nhất 8765.

Đánh giá và cải tiến thời gian tính toán:

Ta nhận thấy, với cách tiếp cận ở trên, mỗi lần xóa, lại phải thực hiện tìm kiếm và so sánh các kí tự từ trái qua phải. Điều này làm tăng thời gian tính toán với số phép tính so sánh khoảng $n \times k$.

Có thể cải tiến điều này với nhận xét:

Giả sử ta xóa chữ số a_i tức là $a_1 \geq a_2 \geq .. \geq a_i < a_{i+1}$. Sau khi xóa a_i xong, để tìm chữ số xóa tiếp theo, ta bắt đầu từ chữ số a_{i-1} và so sánh với chữ số kề phải của nó. Ta có thể sử dụng LifoQueue() hoặc deque() để thực hiện điều này.

Chương trình 1 – Sử dụng LifoQueue

```
import sys, os, argparse, queue
my_stack = queue.LifoQueue()
X  = input("Nhap so: ")
k  = int(input("Nhap k: "))
n  = len(X)
i  = k1 = 0
while i < n:
   if my_stack.empty() or k1 == k:
      my_stack.put(X[i])
      i  += 1
   else:
      top = my_stack.get()
      if top < X[i]:      #Xoa top
         k1 += 1
      else:
         my_stack.put(top)
         my_stack.put(X[i])
         i += 1
while k1 < k:
   my_stack.get()
   k1 += 1
kq = ''
while not my_stack.empty():
   x = my_stack.get()
   kq = x + kq
print("So lon nhat sau khi xoa:",kq)
```

Chương trình 2 – Sử dụng deque

```
import sys, os, argparse, queue
my_deque = queue.deque()
X  = input("Nhap so: ")
k  = int(input("Nhap k: "))
n  = len(X)
i  = k1 = 0
while i < n:
   if len(my_deque) == 0 or k1 == k:
      my_deque.append(X[i])
      i  += 1
   else:
      top = my_deque.pop()
      if top < X[i]:      #Xoa top
         k1 += 1
      else:
         my_deque.append(top)
         my_deque.append(X[i])
         i += 1
while k1 < k:
   my_deque.pop()
   k1 += 1
print("So lon nhat sau khi xoa:")
while len(my_deque) > 0:
   x = my_deque.popleft()
   print(x,end = '')
```

Bài tập 6. Giá trị lớn nhất trong đoạn gồm k số hạng liên tiếp của dãy số

Ý tưởng thuật toán

Thuật toán 1 - Duyệt tuần tự:

Ta có thể xét từng đoạn gồm k số hạng liên tiếp, với mỗi đoạn như vậy, ta sẽ tìm giá trị lớn nhất của các số hạng trong đoạn đó.

Chương trình 1

```
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
print("Nhap",n,"phan tu:")
a = [int(input()) for i in range(n)]
for i in range(n-k+1):
    print(max(a[i:i+k]))
```

❖ ***Nhận xét***

Python cho phép viết chương trình trên thật ngắn gọn và đẹp. Để lấy dãy gồm k số hạng liên tiếp $a_i, a_{i+1}, .., a_{i+k-1}$, ta thực hiện câu lệnh $a[i: i+k]$. Để tìm giá trị lớn nhất của dãy này, ta thực hiện câu lệnh $max(a[i: i+k])$.

Ở đây ta đề cập đến thời gian tính toán (số phép tính cần thực hiện).

Phép toán $max(a[i: i+k])$ được thực hiện bằng cách duyệt tuần tự k số trong dãy để tìm giá trị lớn nhất. Do vậy số phép tính cho thuật toán trên khoảng $(n-k) \times k$. Thuật toán sẽ thực hiện rất chậm khi n và k lớn.

Thuật toán 2 – Sử dụng deque:

Thuật toán sau tốt hơn rất nhiều so với thuật toán 1. Thuật toán sử dụng cấu trúc dữ liệu deque như sau:

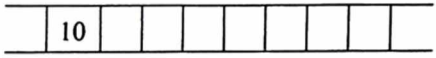
Xét dãy gồm 8 số hạng: 10, 12, 3, 2, 6, 5, 4, 3, k = 3;

Duyệt lần lượt các số hạng của dãy từ a_0 đến a_{n-1}, đưa chúng vào deque như sau:

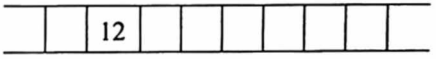
10	12	3	2	6	5	4	3
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

Xét k số hạng đầu tiên:

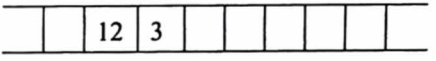
Xét $a_0 = 10 \rightarrow$ append(10)



Xét $a_1 = 12 \rightarrow$ Lấy 10 ra khỏi deque (pop()) vì $12 > 10; \rightarrow$ append(12)

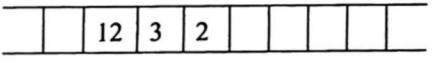


Xét $a_2 = 3, \rightarrow 12 > 3 \rightarrow$ append(3)



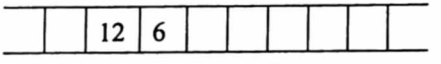
Như vậy, giá trị lớn nhất của $[a_0, a_1, a_2]$ bằng 12.

Xét $a_3 = 2, \rightarrow 3 > 2 \rightarrow$ append(2)

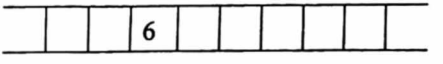


Giá trị lớn nhất của $[a_1, a_2, a_3]$ bằng 12.

Xét $a_4 = 6 \rightarrow 2 < 6 \rightarrow$ pop(); $6 > 3 \rightarrow$ pop(); $12 > 6 \rightarrow$ append(6).

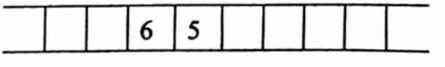


Mà do $12 = a_1$ không thuộc dãy $[a_2, a_3, a_4]$ nên lấy 12 ra khỏi deque.



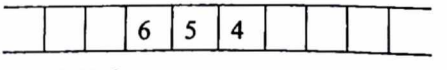
Giá trị lớn nhất của dãy $[a_2, a_3, a_4]$ là 6.

Xét $a_5 = 5 \rightarrow 6 > 5 \rightarrow$ append(5)



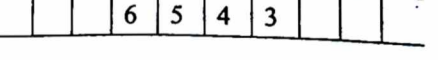
Giá trị lớn nhất của dãy $[a_3, a_4, a_5]$ là 6.

Xét $a_6 = 4 \rightarrow 5 > 4 \rightarrow$ append(4)



Giá trị lớn nhất của dãy $[a_4, a_5, a_6]$ là 6.

Xét $a_7 = 3 \rightarrow 4 > 3 \rightarrow$ append(3)



Mà do 6 = a_4 không thuộc dãy [a_5, a_6, a_7] nên lấy 6 ra khỏi deque.

				5	4	3			

Giá trị lớn nhất của dãy [a_5, a_6, a_7] là 5.

Như vậy, các phần tử trong deque luôn giảm dần. Mỗi khi xét phần tử a_i, nếu phần tử bên phải nhỏ hơn a_i thì ta lấy ra cho đến khi a_i nhỏ hơn phần tử bên phải (hoặc cho đến khi deque rỗng), sau đó append(a_i). Tiếp theo, nếu phần tử ở bên trái deque không thuộc dãy [a_{i-k+1}, .., a_i] thì popleft().

Chương trình 2

```
import sys, os, argparse, queue
my_deque = queue.deque()
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
print("Nhap",n,"phan tu:")
a = [int(input()) for i in range(n)]
for i in range(k):
   while len(my_deque) > 0:
      x = my_deque.pop()
      if x[1] > a[i]:
         my_deque.append(x)
         break;
   my_deque.append((i,a[i]))
print(my_deque[0][1])
for i in range(k, n):
   while len(my_deque) > 0:
      x = my_deque.pop()
      if x[1] > a[i]:
         my_deque.append(x)
         break;
```

```
        my_deque.append((i,a[i]))
        while my_deque[0][0] <= i - k:
            my_deque.popleft()
        print(my_deque[0][1])
```

Bài tập 7. Trò chơi "Bỏ vào - Lấy ra"

Ý tưởng thuật toán

Đây là bài toán có sự thay đổi dữ liệu sau mỗi lần thực hiện (tập số S thay đổi), do vậy, cần sử dụng cấu trúc dữ liệu hàng đợi ưu tiên để mỗi lần lấy phần tử ra, thêm phần tử vào được thực hiện nhanh chóng.

- Bỏ s_0, s_1, .., s_{n-1} vào một PriorityQueue.

- Thực hiện lặp k lần, mỗi lần gồm:

o x = PriorityQueue.get()

o PriorityQueue.put(2*x)

Chương trình

```
import sys, os, argparse, queue
my_PriorityQueue = queue.PriorityQueue()
n = int(input("Nhap n = "))
k = int(input("Nhap k = "))
print("Nhap",n,"so:")
s = [int(input()) for i in range(n)]
for x in s:
    my_PriorityQueue.put(x)
for i in range(k):
    x = my_PriorityQueue.get()
    print(x)
    my_PriorityQueue.put(2*x)
```

Bài tập 8. Truy vấn trung vị

Ý tưởng thuật toán

Sử dụng hai hàng đợi ưu tiên, hàng đợi:

MinPriorityQueue, mỗi lần lấy ra sẽ lấy phần tử nhỏ nhất.

MaxPriorityQueue, mỗi lần lấy ra sẽ lấy phần tử lớn nhất.

Tại lần thứ i ta có 2i + 1 số, giả sử đã sắp xếp tăng thành dãy a_0, a_1, .., a_{2i+1}.

Bỏ i + 1 số a_0, a_1, .., a_i vào MaxPriorityQueue và bỏ i số a_{i+1}, .., a_{2i+1} vào MinPriorityQueue.

Như vậy trung vị sẽ là a_i và là MaxPriorityQueue.get().

Ban đầu MaxPriorityQueue.put(a_0) và MinPriorityQueue rỗng.

Tại lần thứ i, khi thêm hai số x, y, trong 4 số x, y, a_i và a_{i+1} sẽ có hai số nhỏ hơn được đưa vào MaxPriorityQueue và hai số lớn hơn được đưa vào MinPriorityQueue.

Sau khi thêm x, y: Trung vị sẽ bằng MaxPriorityQueue.get().

Để thực hiện MaxPriorityQueue, ta đổi dấu các phần tử trong hàng đợi lúc thêm vào, khi lấy ra, sẽ đổi dấu lại.

Chương trình

```
import sys, os, argparse, queue
MinPriorityQueue = queue.PriorityQueue()
MaxPriorityQueue = queue.PriorityQueue()
a_0 = int(input("Nhap a_0 = "))
MaxPriorityQueue.put(-a_0)
n = int(input("Nhap so lan thuc hien: "))
a_i = -MaxPriorityQueue.get()
for i in range(1, n+1):
   x = int(input("x = "))
   y = int(input("y = "))
```

```
MinPriorityQueue.put(x)
MinPriorityQueue.put(y)
MinPriorityQueue.put(a_i)
u = MinPriorityQueue.get()
v = MinPriorityQueue.get()
MaxPriorityQueue.put(-u)
MaxPriorityQueue.put(-v)
a_i = -MaxPriorityQueue.get()
print("Trung vi lan",i,"bang",a_i)
```

CHỦ ĐỀ 21. XỬ LÝ NGOẠI LỆ - EXCEPTIONS

A. KIẾN THỨC CƠ BẢN

Những lỗi sinh ra trong quá trình thực hiện chương trình được gọi là ngoại lệ (exceptions). Ngoại lệ thường xảy ra trong những trường hợp đặc biệt về dữ liệu hoặc điều kiện gì đó mà trong quá trình thiết kế thuật toán hay viết chương trình, người lập trình chưa lường hết được. Khi gặp lỗi này, thường chương trình sẽ dừng ngay lập tức. Điều này có thể dẫn đến hệ thống không nhận được kết quả như mong đợi hoặc rối loạn hệ thống. Python cũng giống như nhiều ngôn ngữ lập trình khác, cho phép xử lý những trường hợp như vậy nếu xảy ra.

1. Câu lệnh try ... except

Cú pháp

```
try:
   <Khối lệnh 1>
except:
   <Khối lệnh 2>
```

Ý nghĩa thực hiện

Thực hiện **<Khối lệnh 1>**. Nếu trong quá trình thực hiện **<Khối lệnh 1>**, có phát sinh lỗi thì **<Khối lệnh 2>** sẽ thực hiện.

Ví dụ 1. Chương trình:

```
a = 10
b = 2
try:
   c = a/b
   print("Thuong a/b: ",c)
except:
   print("Gap loi chia 0")
print("Chuong trinh da ket thuc")
```

☞ Kết quả:

```
Thuong a/b:  5.0
Chuong trinh da ket thuc
```

Ví dụ 2. Chương trình:

```
a = 10
b = 0
try:
   c = a/b
   print("Thuong a/b: ",c)
except:
   print("Gap loi chia 0")
print("Chuong trinh da ket thuc")
```

☞ Kết quả:

```
Gap loi chia 0
Chuong trinh da ket thuc
```

❖ ***Nhận xét***

Ta nhận thấy, trong chương trình 2, khi b = 0, câu lệnh a/b sẽ gặp lỗi, lúc này câu lệnh `print("Gap loi chia 0")` *được thực hiện. Điều quan trọng nữa là chương trình được thực hiện đầy đủ, không kết thúc khi gặp lỗi.*

Còn nếu thực hiện chương trình:

```
a = 10
b = 0
c = a/b
print("Thuong a/b: ",c)
print("Chuong trinh da ket thuc")
```

Thì chương trình sẽ dừng lại và kết thúc khi thực hiện đến câu lệnh c = a/b.

2. Câu lệnh try ... except ... else

Cú pháp

```
try:
   <Khối lệnh 1>
except:
   <Khối lệnh 2>
else:
   <Khối lệnh 3>
```

Ý nghĩa thực hiện

Thực hiện **<Khối lệnh 1>**. Nếu trong quá trình thực hiện **<Khối lệnh 1>**, có phát sinh lỗi thì **<Khối lệnh 2>** sẽ thực hiện. Nếu **<Khối lệnh 1>** không phát sinh lỗi thì **<Khối lệnh 3>** sẽ thực hiện.

Ví dụ 3. Chương trình:

```
a = 10
b = 2
try:
   c = a/b
except:
   print("Gap loi chia 0")
else:
   print("Thuong a/b: ",c)
print("Chuong trinh da ket thuc")
```

☞ Kết quả:

```
Thuong a/b:  5.0
Chuong trinh da ket thuc
```

Ví dụ 4. Chương trình:

```
a = 10
b = 0
```

```
try:
    c = a/b
except:
    print("Gap loi chia 0")
else:
    print("Thuong a/b: ",c)
print("Chuong trinh da ket thuc")
```

☞ Kết quả:

```
Gap loi chia 0
Chuong trinh da ket thuc
```

3. Câu lệnh try ... except ... else ... finally,

Cú pháp

```
try:
    <Khối lệnh 1>
except:
    <Khối lệnh 2>
else:
    <Khối lệnh 3>
finally:
    <Khối lệnh 4>
```

Ý nghĩa thực hiện

Khi thực hiện các khối lệnh trên, `<Khối lệnh 4>` luôn được thực hiện.

Ví dụ 5. Chương trình:

```
a = 10
b = 2
```

```
try:
    c = a/b
except:
    print("Gap loi chia 0")
else:
    print("Thuong a/b: ",c)
finally:
    print("Khoi lenh try ... finally da thuc hien")
print("Chuong trinh da ket thuc")
```

☞ Kết quả:

```
Thuong a/b:  5.0
Khoi lenh try ... finally da thuc hien
Chuong trinh da ket thuc
```

4. Các lớp exceptions

Ở trên, ta mới đề cập đến lỗi chia 0, `ZeroDivisionError`. Ngoài ra, còn nhiều lỗi khác có thể sinh ra trong khi thực hiện chương trình. Python cung cấp một lớp cơ sở của tất cả các exception đó là lớp **`Exception`**.

Ví dụ 6. Chương trình:

```
a = 10
b = 0
try:
    u = a/b
except Exception as e:
    print(type(e), e)
```

☞ Kết quả:

```
<class 'ZeroDivisionError'> division by zero
```

Ví dụ 7. Chương trình:

```
List = [1, 2, 3]
try:
    u = List[3]
except Exception as e:
    print(type(e), e)
```

☞ Kết quả:

```
<class 'IndexError'> list index out of range
```

Một số lớp exception thường gặp

Tên lớp	Lỗi xuất hiện khi	Ví dụ minh họa

ImportError	import sai tên, phương thức trong một modunle hay package.	try:
		  from sys import Python
		except Exception as e:
		  print(type(e), e)

IndexError	Truy cập đến chỉ số không thuộc danh sách, string, tuple,...	List = [0, 1, 2]
		try:
		  u = List[3]
		except Exception as e:
		  print(type(e), e)

KeyError	Truy cập đến phần tử có khóa không thuộc từ điển.	Dict = {1:"A", 2:"B"}
		try:
		  u = Dict[3]
		except Exception as e:
		  print(type(e), e)

NameError	Sử dụng biến có tên chưa xác định.	try:
		  u = v
		except Exception as e:
		  print(type(e), e)

TypeError	Tính toán trên các đối tượng có kiểu không phù hợp.	try:
		  u = 12/"12"
		except Exception as e:
		  print(type(e), e)

ValueError	Tính toán trên các đối tượng phù hợp nhưng giá trị không phù hợp	try:
		  u = int("a")
		except Exception as e:
		  print(type(e), e)

Chương trình

```
while 1 > 0:
   try:
      a = int(input("Nhap so nguyen a: "))
      b = int(input("Nhap so nguyen b: "))
      u = a/b
   except ZeroDivisionError:
      print("Gia tri b = 0")
      print("Ban hay nhap lai:")
   except ValueError:
      print("Gia tri nhap vao khong phai so")
      print("Ban hay nhap lai:")
   else:
      break
print("a/b = ", u)
```

Bài tập 2. Đọc tệp

Ý tưởng thuật toán

Sử dụng ngoại lệ `FileNotFoundError` để nhận biết ngoại lệ mở tệp không tồn tại để đọc dữ liệu.

Chương trình

```
dr = input("Nhap tep de doc du lieu:")
try:
   f = open(dr,"r")
except FileNotFoundError:
   print("Khong ton tai tep vua nhap")
else:
   g = open("C:\Copy.txt","w")
   st = f.read()
   g.write(st)
   f.close()
   g.close()
```

CHỦ ĐỀ 22. ĐỒ HỌA CÙNG VỚI TKINTER

A. KIẾN THỨC CƠ BẢN

Python cung cấp package Tkinter cho phép lập trình ở chế độ cửa sổ, đồ họa rất thuận tiện.

Để sử dụng package Tkinter, ta thực hiện:

```
import tkinter
from tkinter import *
```

1. Tạo cửa sổ đồ họa

```
import tkinter
from tkinter import *
<Tên của sổ> = Tk()
<Tên của sổ>.title([tiêu đề cửa sổ])
<Tên của sổ>.mainloop()
```

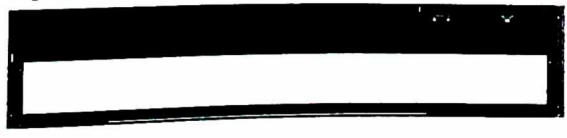
Thiết lập kích thước cửa sổ

<Tên cửa sổ>.**geometry**("cột ×dòng")

Ví dụ 1. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x30")
my_window.mainloop()
```

Kết quả:



2. Đối tượng Label

Đối tượng label (nhãn) dùng để hiển thị một văn bản.

Khai báo

<Tên nhãn> = **Label**(<Tên cửa sổ> , **text** = <Nội dung hiển thị>)

Vị trí hiển thị

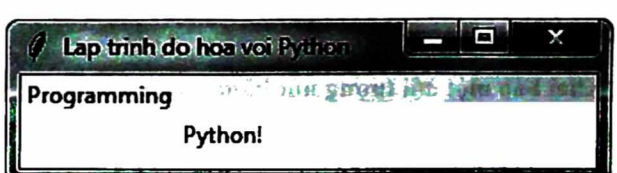
<Tên nhãn>.**grid**(**column** = x, **row** = y)

Nhãn sẽ hiển thị tại vị trí cột x và dòng y.

Ví dụ 2. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Programming")
lb1.grid(column = 0, row = 0)
lb2 = Label(my_window, text = "Python!")
lb2.grid(column = 20, row = 4)
my_window.mainloop()
```

Kết quả:



Thiết lập Font và cỡ chữ

<Tên nhãn> = **Label**(<Tên cửa sổ>, **text** = <Nội dung hiển thị>, **font** = (<Tên font>, <Cỡ font>))

Ví dụ 3. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Programming")
lb1.grid(column = 0, row = 0)
lb2 = Label(my_window, text = "Python!")
lb2 = Label(my_window, text = "Python!", font
=("Arial Bold", 16))
lb2.grid(column = 20, row = 4)
my_window.mainloop()
```

☞ Kết quả:

3. Đối tượng Button

Khai báo một đối tượng nút bấm - Button

<Tên nút bấm> = **Button**(<Tên cửa sổ>, **text** = <Nội dung hiển thị>)

Vị trí hiển thị

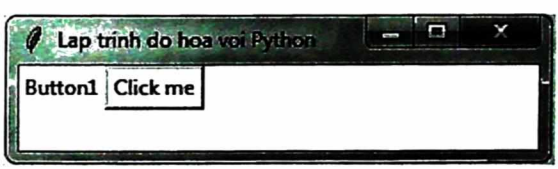
<Tên nút bấm>.**grid**(column = x, row = y)

Nút bấm sẽ hiển thị tại cột x và dòng y.

Ví dụ 4. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Button1")
lb1.grid(column = 0, row = 0)
bt1 = Button(my_window, text = "Click me")
bt1.grid(column = 2, row = 0)
my_window.mainloop()
```

☞ Kết quả:



Thiết lập màu với nút bấm button

<Tên nút bấm> = **Button**(<Tên cửa sổ>, **text**=<Nội dung hiển thị>, **bg** = <Màu nền>, **fg** =<Màu nội dung hiển thị>)

Ví dụ 5. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Button1")
lb1.grid(column = 0, row = 0)
```

```
bt1 = Button(my_window,text="Click me", bg="orange",
fg= "red")
bt1.grid(column = 2, row = 0)
my_window.mainloop()
```

☞ Kết quả:

Xử lý thông tin khi nút bấm được bấm

Mỗi khi bấm một nút bấm, chương trình sẽ thực hiện một công việc nào đó. Công việc đó được tổ chức dưới dạng một chương trình con.

def <Clicked>():

<Khối lệnh>

Thiết lập thông số cho nút bấm.

<Tên nút bấm> = **Button**(<Tên cửa sổ>, **text** = <Nội dung hiển thị>, **command** = <Clicked>)

Ví dụ 6. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Button1")
lb1.grid(column = 0, row = 0)
def bt1_clicked():
   lb1.configure(text="Button1 was clicked !!")
bt1 = Button(my_window,text="Click me", command =
bt1_clicked)
```

```
bt1.grid(column = 2, row = 0)
my_window.mainloop()
```

☞ Kết quả:

Trước khi bấm:

Sau khi bấm:

4. Đối tượng textbox

Đối tượng textbox dùng để nhập thông tin.

Tạo một textbox

<Tên textbox> = **Entry**(<Tên cửa sổ>, **width** = <Độ rộng>)

Lấy thông tin từ textbox

<Tên textbox>.**get()**

Ví dụ 7. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lb1 = Label(my_window, text = "Entry1:")
lb1.grid(column = 0, row = 0)
```

```
txt = Entry(my_window, width = 20)
txt.grid(column = 1, row = 0)
def bt1_clicked():
    lb1.configure(text="Entry1 = "+txt.get())
bt1 = Button(my_window,text="Click me", command =
bt1_ clicked)
bt1.grid(column = 2, row = 0)
my_window.mainloop()
```

☞ Kết quả:

Trước khi bấm: Nhập Python vào textbox.

Sau khi bấm:

5. Đối tượng Combobox

Đối tượng Combobox cho phép hiển thị danh sách các đề mục, các lựa chọn, người sử dụng có thể lựa các đề mục từ danh sách này.

Để sử dụng đối tượng Combobox, ta sử dụng lớp **ttk** của **tkinter**.

```
import tkinter
from tkinter import *
from tkinter.ttk import *
```

- Khai báo đối tượng Combobox

<Tên Combo> = **Combobox**(<Tên cửa sổ>)

- Thiết lập danh sách các đề mục, các lựa chọn

<Tên Combo>[**'values'**] = (<đề mục 0>, <đề mục 1>, .. , <đề mục n>)

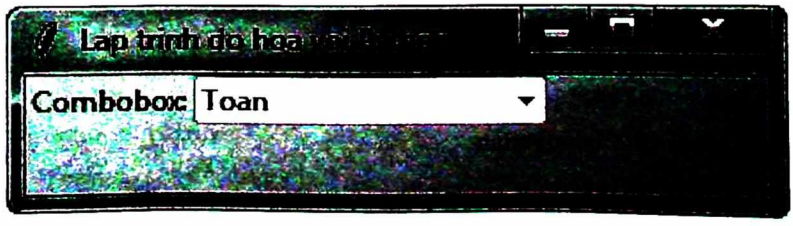
- Lấy danh đề mục được lựa chọn

<Tên Combo>.**get()**

Ví dụ 8. Chương trình:

```
import tkinter
from tkinter import *
from tkinter.ttk import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
lab1 = Label(my_window, text = "Combobox:")
lab1.grid(column = 0, row = 0)
combo = Combobox(my_window)
combo['values'] = ('Toan', 'Tin', 1, 2)
combo.current(0)  #chon de muc 0
combo.grid(column = 1, row = 0)
my_window.mainloop()
```

☞ Kết quả:



5. Đối tượng Checkbutton

Khai báo

<Tên Checkbutton> = **Checkbutton**(<Tên cửa sổ>, **text** = <Nhãn>)

Ví dụ 9. Chương trình:

```
import tkinter
from tkinter import *
from tkinter.ttk import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
chk_state = BooleanVar()
chk_state.set(True)
chk = Checkbutton(my_window, text = "Choose",
var = chk_state)
chk.grid(column = 0, row = 0)
my_window.mainloop()
```

Kết quả:

7. Đối tượng Radiobutton

Khai báo

<Tên radio> = **Radiobutton**(<Tên cửa sổ>, **text** = <Nhãn>, **value** = <Giá trị>)

Ví dụ 10. Chương trình:

```
import tkinter
from tkinter import *
```

```
from tkinter.ttk import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x50")
rad1 = Radiobutton(my_window,text='Radiobutton 1',
value=1)
rad2 = Radiobutton(my_window,text='Radiobutton 2',
value=2)
rad1.grid(column=0, row=0)
rad2.grid(column=0, row=1)
my_window.mainloop()
```

☞ Kết quả:

8. Đối tượng ScrolledText

Đối tượng ScrolledText dùng để nhập đoạn văn bản. Cửa sổ của ScrolledText cho phép cuộn lên – xuống để xem nội dung.

- Tạo đối tượng **ScrolledText**

```
from tkinter import scrolledtext
```

<Tên Scroll> = **scrolledtext. Scrolledtext(**<Tên cửa sổ>, **width** = x, **height** = y)

Tạo một `scrolledtext` có chiều rộng x và chiều cao y.

Ví dụ 11. Chương trình:

```
import tkinter
from tkinter import *
from tkinter import scrolledtext
```

```
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x120")
lab1 = Label(my_window,text = "ScrolledText: ")
lab1.grid(column=0,row=0)
txt =
scrolledtext.ScrolledText(my_window,width=25,height=5)
txt.grid(column=1,row=1)
my_window.mainloop()
```

☞ Kết quả:

➧ Thiết lập nội dung trong `scrolledtext`

<Tên Scroll>.**insert(INSERT,**<Nội dung>)

Ví dụ 12. Chương trình:

```
import tkinter
from tkinter import *
from tkinter import scrolledtext
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x120")
lab1 = Label(my_window,text = "ScrolledText: ")
lab1.grid(column=0,row=0)
```

```
txt =
scrolledtext.ScrolledText(my_window,width=25,heigh
t=5)
txt.grid(column=1,row=1)
txt.insert(INSERT,"Programming Python.\n")
txt.insert(INSERT,"Programming Python.\n")
txt.insert(INSERT,"Programming Python.\n")
my_window.mainloop()
```

☞ Kết quả:

➧ **Xóa nội dung trong `scrolledtext`**

<Tên Scroll>.delete(1.0, END)

Xóa tất cả nội dung trong `scrolledtext`.

9. Đối tượng MessageBox

Đối tượng MessageBox dùng để hiển thị một thông báo.

➧ Tạo đối tượng MessageBox

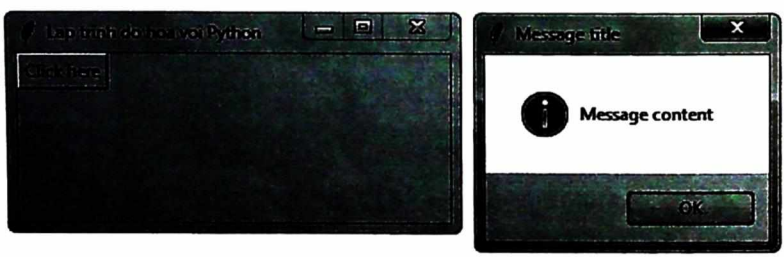
```
from tkinter import messagebox
messagebox.showinfo("Tiêu đề", "Nội dung")
```

Ví dụ 13. Chương trình:

```
import tkinter
from tkinter import *
from tkinter import messagebox
```

```
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x120")
def clicked():
    messagebox.showinfo('Message     title','Message
content')
btn=Button(my_window,text='Click   here',command   =
clicked)
btn.grid(column=0,row=0)
my_window.mainloop()
```

☞ Kết quả:



Các phương thức khác của đối tượng `messagebox`

Phương thức:

`messagebox.showwarning`("Message title", "Message warning")

Phương thức:

messagebox.showerror("Message title", "Message error")

Phương thức:

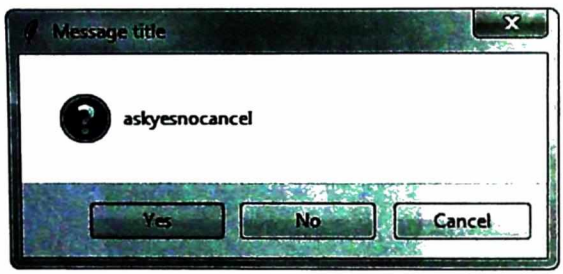
res = messagebox.askquestion("Message title","Message askquestion")

Phương thức:

res = messagebox.askyesno("Message title","Message askyesno")

Phương thức:

`res = messagebox.askyesnocancel`("Message title", 'askyesnocancel")

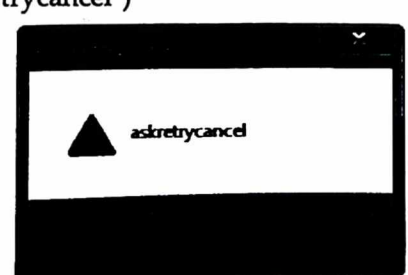


Phương thức:

`res = messagebox.askokcancel`("Message title","Message askokcancel")

Phương thức:

`res = messagebox.askretrycancel`("Message title","askretrycancel")



❖ ***Chú ý***

Giá trị **res** = True *khi bấm nút* Yes, Ok, Retry.

Giá trị **res** = False *khi bấm nút* No, Cancel.

Riêng `res = messagebox.askyesnocancel` *sẽ trả về* True, False, None *khi bấm* Yes, No, Cancel.

10. Đối tượng SpinBox

Tạo đối tượng SpinBox

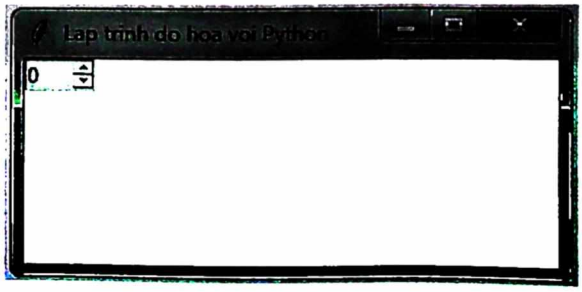
<Tên Spin> = **Spintbox**(<Tên cửa sổ>, **from_** x, **to_** y, **width** h)

<Tên Spin> sẽ nhận giá trị là một số từ x đến y, độ rộng h.

Ví dụ 14. Chương trình:

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x120")
spin = Spinbox(my_window, from_ = 0,to_ = 10, width = 4)
spin.grid(column=0,row=0)
my_window.mainloop()
```

☞ Kết quả:



- Thiết lập các giá trị cho Spinbox

Ngoài việc thiết lập giá trị của Spinbox thuộc dải từ x đến y, ta còn có cách thiệt lập tập các giá trị rời rạc

<Tên Spin> = **Spintbox**(<Tên cửa sổ>, **values = (3, 8, 11), width** 4)

- Thiết lập giá trị mặc định cho Spinbox

```
var = IntVar()
var.set(36)                    #Giá trị mặc định là 36
```

<Tên Spin> = **Spintbox**(<Tên cửa sổ>, `from_ = 0, to_ = 100, width` 4, **textvariable=var**)

11. Đối tượng Progressbar

- Tạo đối tượng Progressbar

```
import tkinter
from tkinter import *
from tkinter.ttk import Progressbar
```

<Tên Progres> = **Progressbar**(<Tên cửa sổ>, **length** = <Độ dài>)

- Khởi tạo giá trị cho Progressbar

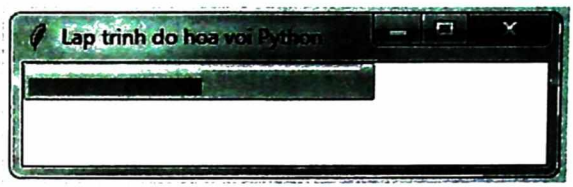
<Tên Progres>["value"] = <Giá trị>

<Giá trị> ∈ [0; 100].

Ví dụ 15. Chương trình:

```
import tkinter
from tkinter import *
from tkinter.ttk import Progressbar
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x60")
bar = Progressbar(my_window, length=200)
bar["value"] = 50
bar.grid(column = 0,row = 0)
my_window.mainloop()
```

☞ Kết quả:

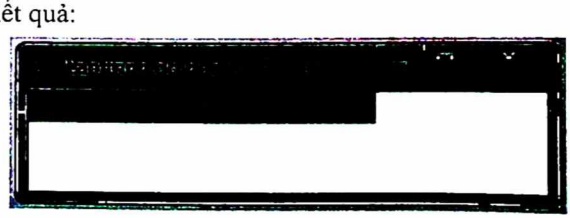


- Thiết lập màu cho `Progressbar`

Ví dụ 16. Chương trình:

```
import tkinter
from tkinter import *
from tkinter.ttk import Progressbar
from tkinter import ttk
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x60")
style = ttk.Style()
style.theme_use('default')
style.configure("black.Horizontal.TProgressbar",
background='black')
bar = Progressbar(my_window, length=200,
style='black.Horizontal.TProgressbar')
bar["value"] = 50
bar.grid(column = 0,row = 0)
my_window.mainloop()
```

☞ Kết quả:



12. Đối tượng filedialog

- Tạo đối tượng filedialog

```
from tkinter import filedialog
```

<Tên filedialog> = filedialog.askopenfilename()

Tạo hộp thoại mở tệp.

- Một số phương thức khác của đối tượng filedialog

```
filedialog.askopenfilenames()
filedialog.askopenfilename(filetypes = (("Text
files","*.txt"),("all files","*.*")))
filedialog.askdirectory()
```

13. Đối tượng menu

- Tạo đối tượng menu

```
from tkinter import Menu
<Tên menu> = Menu(Tên cửa sổ)
<Tên menu>.add_cascade(label= <Tên nhãn>)
```

Ví dụ 17. Chương trình:

```
import tkinter
from tkinter import *
from tkinter import Menu
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x60")
my_menu = Menu(my_window)
my_menu.add_cascade(label="File")
my_menu.add_cascade(label="Edit")
my_window.config(menu=my_menu)
my_window.mainloop()
```

☞ Kết quả:

➧ Thêm các thành phần vào menu

```
import tkinter
from tkinter import *
from tkinter import Menu
my_window = Tk()
my_window.title("Lap trinh do hoa voi Python")
my_window.geometry("300x60")
my_menu = Menu(my_window)
new_item1 = Menu(my_menu)
new_item2 = Menu(my_menu)
new_item1.add_command(label='Save')
new_item1.add_separator()
new_item1.add_command(label='Save as')
new_item2.add_command(label='Copy')
my_menu.add_cascade(label="File", menu =
new_item1)
my_menu.add_cascade(label="Edit", menu =
new_item2)
my_window.config(menu=my_menu)
my_window.mainloop()
```

☞ Kết quả:

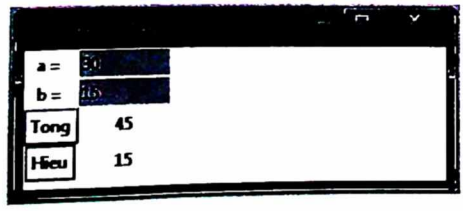
➧ Cài đặt lệnh khi chọn các thành phần trong menu

```
def my_Clicked():
    print("Save is chosen")
new_item1.add_command(label='Save', command =
my_Clicked)
```

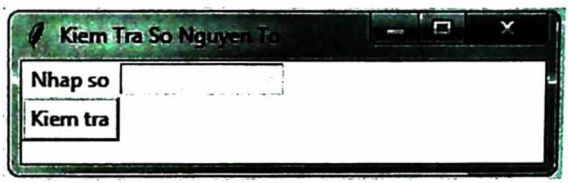
B. BÀI TẬP ÔN LUYỆN

Bài tập 1. *Hãy thiết kế một chương trình có giao diện đồ họa như hình vẽ và có chức năng tính tổng, hiệu của hai số nguyên.*

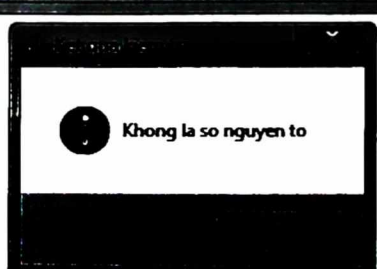
Khi nhập giá trị a và b, bấm vào các nút lệnh Tong, Hieu thì ta được kết quả.



Bài tập 2. *Hãy thiết kế một chương trình có giao diện đồ họa như hình vẽ và có chức năng nhập vào một số nguyên dương và kiểm tra số nhập vào có phải là số nguyên tố hay không?*



Khi nhập số và bấm vào nút lệnh thì ta có kết quả kiểm tra.



Bài tập 3. Chọn số trên lưới ô vuông

Hãy thiết kế chương trình có giao diện đồ họa như hình vẽ:

Gồm lưới ô vuông, gồm 8 hàng và 8 cột. Mỗi ô vuông tương ứng là một Button. Trên mỗi ô vuông có hiển thị một số nguyên ngẫu nhiên từ 0 đến 10.

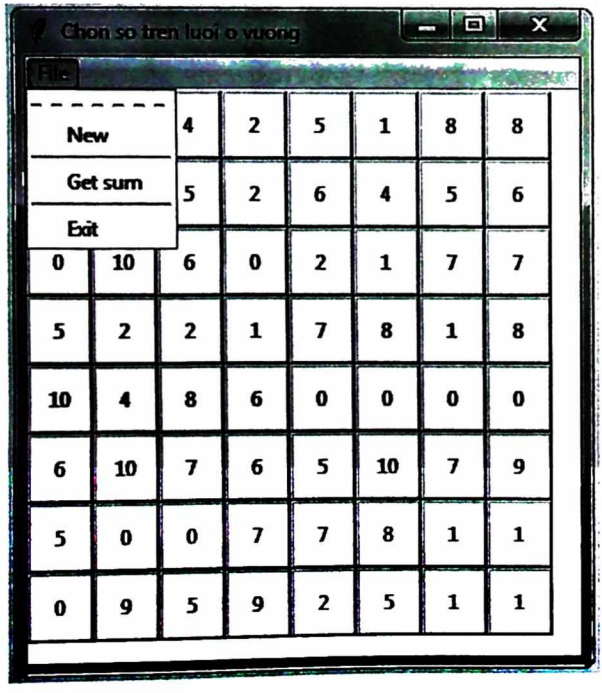
Khi mỗi ô vuông (Button) được bấm, chúng chuyển sang màu vàng và bị khóa (disabled) không cho phép bấm lần khác.

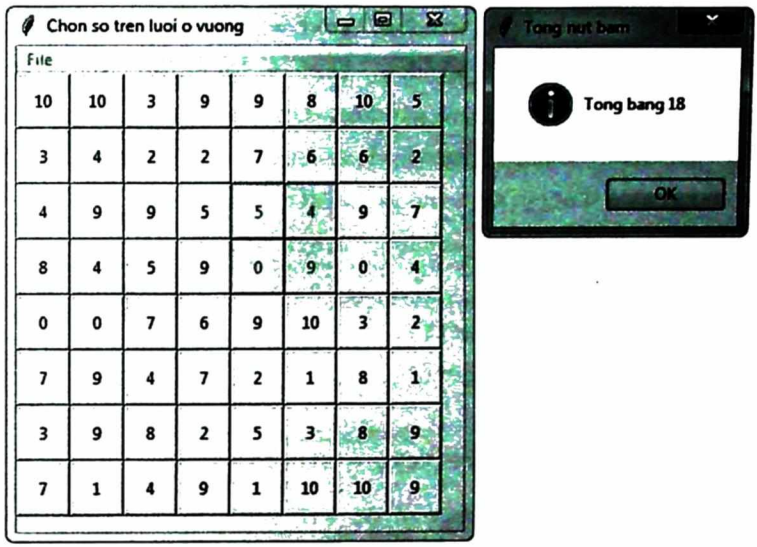
Có một menu File gồm 3 lựa chọn:

- Lựa chọn New → Tạo mới lại lưới ô vuông.

- Lựa chọn Get sum → Hiển thị tổng các số trên các ô vuông được bấm.

- Lựa chọn Exit → Thoát khỏi của sổ đồ họa.





C. THUẬT TOÁN VÀ HƯỚNG DẪN GIẢI

Bài tập 1

Ý tưởng thuật toán

Sử dụng 4 nhãn **Lable** với text là: "a = ", "b = " và hai nhãn còn lại dùng để hiển thị kết quả tính tổng và hiệu, có text khởi tạo là chuỗi rỗng.

Sử dụng hai **Entry** để chứa hai số nhập được nhập vào.

Để lấy giá trị được nhập vào, ta thực hiện:

<Biến Entry>.**get()**

Sử dụng hai nút lệnh **Button** để thực hiện hai phép toán tổng và hiệu. Với mỗi Button, xây dựng hàm thực hiện tính tổng, hiệu khi nút lệnh được bấm. **Chương trình**

```
import tkinter
from tkinter import *
my_window = Tk()
my_window.title("Tong va Hieu")
my_window.geometry("300x100")
```

```
lba   = Label(my_window, text = "a = ")
lbb   = Label(my_window, text = "b = ")
lbsum    = Label(my_window, text = "")
lbsub    = Label(my_window, text = "")
txta  = Entry(my_window, width = 10)
txtb  = Entry(my_window, width = 10)
def getsum():
   a = int(txta.get())
   b = int(txtb.get())
   c = a + b
   lbsum.configure(text = str(c))
def getsub():
   a = int(txta.get())
   b = int(txtb.get())
   c = a - b
   lbsub.configure(text = str(c))
btsum = Button(my_window, text = "Tong", command =
getsum)
btsub = Button(my_window, text = "Hieu", command =
getsub)
lba.grid(column = 0, row = 0)
lbb.grid(column = 0, row = 2)
txta.grid(column = 10, row = 0)
txtb.grid(column = 10, row = 2)
btsum.grid(column = 0, row = 5)
lbsum.grid(column = 10, row = 5)
btsub.grid(column = 0, row = 10)
lbsub.grid(column = 10, row = 10)
my_window.mainloop()
```

Bài tập 2

Ý tưởng thuật toán

Sử dụng một Label với text = "Nhap so", một Entry để nhập số tự nhiên cần kiểm tra tính nguyên tố và một nút lệnh Button với text = "Kiem tra".

Đối với nút lệnh, cần cài đặt hàm `command = isPrime` , trong đó `isPrime()` là hàm kiểm tra tính nguyên tố.

Chương trình

```
import tkinter
from tkinter import *
from tkinter import messagebox
my_window = Tk()
my_window.title("Kiem Tra So Nguyen To")
my_window.geometry("300x60")
lbN   = Label(my_window, text = "Nhap so")
txtN  = Entry(my_window, width = 15)
def isPrime():
   n = int(txtN.get())
   NT = 1
   if n < 2:
      NT = 0
   else:
      i = 2
      while i*i <= n:
         if n % i == 0:
            NT = 0
            break
         i+=1
   if NT == 1:
      messagebox.showinfo('Ket qua kiem tra','La so nguyen to')
```

```
    else:
        messagebox.showinfo('Ket qua kiem tra','Khong la
so nguyen to')
btKt  = Button(my_window, text = "Kiem tra", command =
isPrime)
lbN.grid(column = 1, row = 2)
txtN.grid(column = 10, row = 2)
btKt.grid(column = 1, row = 4)
my_window.mainloop()
```

Bài tập 3

Ý tưởng thuật toán

- Tạo lưới các nút bấm:

Để tạo lưới nút bấm, ta có thể sử dụng một list hai chiều gồm 8 dòng và 8 cột, mà mỗi phần tử của list là một Button. Chẳng hạn `listbutton[8][8]`.

Các Button cần được thiết lập kích thước giống nhau để tạo thành lưới cân đối.

```
listbutton[i][j].config(height = 2, width = 4)
```

- Đổi màu và khóa các button được bấm.

```
listbutton[i][j].config(bg = "yellow", state =
"disabled")
```

- Lấy các số hiển thị trên các button được bấm.

```
listbutton[i][j].cget("text")
```

- Lấy tổng các số trên các button được bấm.

Mỗi nút bấm `listbutton[i][j]` được thiết lập `command = lambda ii = i, jj=j: clicked(ii,jj))`. Chú ý là, ở đây bắt buộc phải sử dụng hàm lambda, nếu ta thiết lập `command = clicked(i,j)` thì mỗi khi chương trình sẽ không hoạt động như mong muốn.

- Thiết lập lại màu và trạng thái cho button mỗi khi chon **New.**

Lấy màu của cửa sổ orig_color = my_window.cget("background")

```
   listbutton[i][j].config(text = str(x),bg =
orig_color, state = "normal" )
```

- Thiết lập chức năng **Exit**

Sử dụng hàm hủy của lớp window.

```
   my_window.destroy()
```

Chương trình

```
from random import randint
import tkinter
from tkinter import *
from tkinter import Menu
from tkinter import messagebox
my_window = Tk()
my_window.title("Chon so tren luoi o vuong")
my_window.geometry("320x340")
my_menu = Menu(my_window)
new_item1 = Menu(my_menu)
sum = 0
listbutton = []
def clicked(i, j):
   global sum
   sum = sum + int(listbutton[i][j].cget("text"))
   listbutton[i][j].config(bg = "yellow", state = "disabled'
def reset():
   global sum
   sum = 0
   orig_color = my_window.cget("background")
   for i in range(8):
      for j in range(8):
         x = randint(0, 10)
```

```
        listbutton[i][j].config(text = str(x),bg =
orig_color, state = "normal" )
def getsum():
   messagebox.showinfo("Tong nut bam","Tong bang
"+str(sum))

def exit():
   my_window.destroy()

new_item1.add_command(label='New', command = reset)
new_item1.add_separator()
new_item1.add_command(label='Get sum', command = getsum)
new_item1.add_separator()
new_item1.add_command(label='Exit', command = exit)
my_menu.add_cascade(label="File", menu = new_item1)
my_window.config(menu=my_menu)
for i in range(8):
   listRow = []
   for j in range(8):
      x = randint(0,10)
      bt = Button(my_window, text = str(x),
         command = lambda ii = i, jj=j: clicked(ii,jj))
      bt.config(height = 2, width = 4)
      bt.grid(column = j*2, row = i)
      listRow.append(bt)
   listbutton.append(listRow)
my_window.mainloop()
```

TÀI LIỆU THAM KHẢO

[1] Richard L. Halterman (2011), *Learning to Program with Python*. Retrieved from https://web.itu.edu.tr/hulyayalcin/MAK230E_PythonProgramming/%5B2011%5D%5BHalterman%5DLEARNINGTOPROGRAMWITHPYTHON.pdf

[2] Dave Kuhlman (2013), *Beginning Python, Advanced Python, and Python Exercises*. Retrieved from https://www.davekuhlman.org/python_book_01.pdf

[3] Mohit., & Bhaskar N. Das (2017), Learn Python In 7 Days, Packt Publishing.

[4] Zed A. Shaw (2014), *Learn Python The Hard Way*, 2014, Third Edition.

[5] Stack Overflow Documentation, Python Notes for Professionals – goalkicer.com. Retrieved from https://goalkicker.com/PythonBook/PythonNotesForProfessionals.pdf

[6] https://www.w3schools.com.

[7] https://www.python.org/

[8] https://realpython.com/

NHÀ XUẤT BẢN
ĐẠI HỌC QUỐC GIA HÀ NỘI
16 Hàng Chuối - Hai Bà Trưng - Hà Nội

Giám đốc - Tổng Biên tập: (024)39715011
Hành chính: (024)39714899; Fax: (024)39724736
Quản lý xuất bản: (024) 39728806
Biên tập: (024) 39714896

Chịu trách nhiệm xuất bản: Giám đốc - Tổng biên tập: TS. PHẠM THỊ TRÂM

Biên tập: TRỊNH THỊ THU HÀ
Chế bản: ĐỖ THỊ HỒNG SÂM
Trình bày bìa: NGUYỄN NGỌC ANH
Đối tác liên kết: Tác giả

SÁCH LIÊN KẾT

ĐƯỜNG VÀO LẬP TRÌNH PYTHON

Mã số: 1L-06ĐH2020
In 1.000 bản, khổ 16x24cm tại Công ty TNHH XS DV Bao bì Kiến Á
Địa chỉ: 320/32A Trần Bình Trọng, phường 4, quận 5, TP. Hồ Chí Minh
Số xác nhận ĐKXB: 620-2020/CXBIPH/08-65/ĐHQGHN, ngày 27/02/2020
Quyết định xuất bản số: 181 LK-TN/QĐ - NXB ĐHQGHN, ngày 03/3/2020
In xong và nộp lưu chiểu năm 2020.